UNIVERSITETET I OSLO
Fysisk Institutt

Kinect-basert autopilot for Quadrokopter

Masteroppgave

Jon Øyvind Trømborg

26. mai 2014

Forord

Denne oppgaven er avslutningen på mitt masterstudie i Elektronikk og Datateknologi, studieretning Kybernetikk, ved Universitetet i Oslo. Jeg vil gjerne takke min veileder, Oddvar Hallingstad, for innspill og veiledning under denne oppgaven. Jeg vil også takke min søster for korrekturlesing og min kjæreste for hennes tålmodig under denne oppgaven.

Jon Øyvind Trømborg

Kjeller 26. mai-14

Sammendrag

I denne oppgaven er det sett på muligheten av å benytte Microsoft Kinect sensor for treghetsnavigasjon for quadrokoptere innendørs. Det har blitt sett på hvilke data som er tilgjengelige fra Kinect sensoren, samt gjort kalibreringer. Deretter har det blitt gjort bildeanalyse for gjenkjenning av quadrokopter, med måleusikkerheter. Videre ble posisjon og orientering av quadrokopter beregnet og det ble laget et grensesnitt mellom PC og fjernkontroll. Til slutt ble det laget en autopilot til denne.

Innholdsfortegnelse

Figurliste

Tabelliste

1 Innledning

1.1 Problemstilling

Dersom det skal lages autopiloter for små UAV'er, er de avhengige av sensorer som måler posisjon foruten gyroer og akselerometre. Utendørs blir dette gjerne løst med GPS, men innendørs må dette løses på andre måter. I denne oppgaven skal det undersøkes om Microsofts Kinect sensor kan brukes til å finne både stilling og posisjon til et lite quadrokopter(QK). Oppgaven består av følgende punkter:

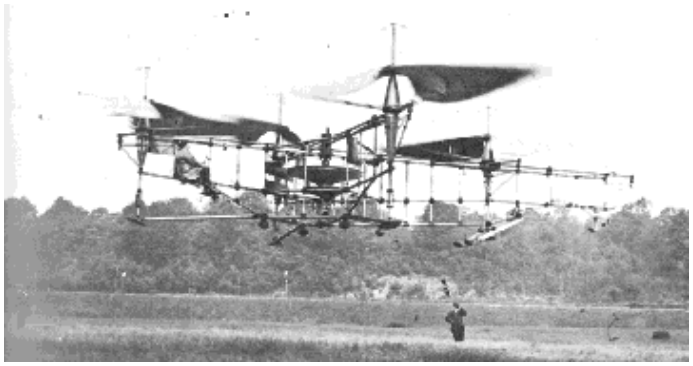
1. Undersøke hvilke data som er tilgjengelige fra Kinect og teknologien bak.
2. Finne metoder for å gjenkjenne et quadrokopter i et bilde.
3. Finne posisjon og orientering av QK i rommet.
4. Lage grensesnitt mellom QK og PC.
5. Lage autopilot basert på de foregående data.

1.2 UAV

En UAV, gjerne kaldt drone, er et flygende fartøy hvor det er ingen operatør i selve fartøyet. UAV`en har egne datamaskiner og sensorer som står for styringen/reguleringen av pådrag og eventuelle styreflater, for å holde den flyvende. En UAV kan være alt fra luftskip til helikopter. UAV'en kan være delvis autonom eller fullstendig autonom. Hos en fullstendig autonom dronen vil oppdraget allerede være lastet inn i dronens datamaskiner. Den vil da, på bakgrunn av oppdraget og annen logikk, ta avgjørelser på egenhånd. Typisk kan dette være overvåkning av områder hvor endringer i bilde vil være av interesse. Delvis autonome UAV`er blir ofte assosiert med amerikaneres Predator- og MQ-9 Reaper droner. Delvis autonome droner flyr av seg selv, men her er det operatører som avgjør hvilke handlinger som dronen skal gjennomføre.

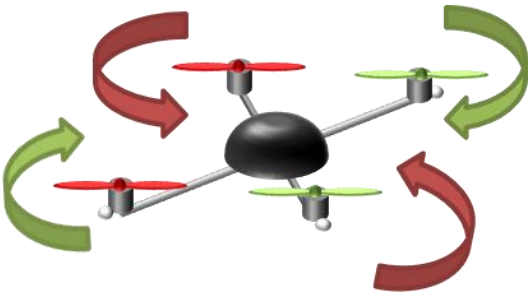
1.2.1 Quadrokopter

Ettersom sensor-teknologien krymper og elektronikken blir billigere, har tilgjengeligheten av droner for allmenheten, og da spesielt 4-rotors helikoptre, eksplodert. Den første vellykkete flyvningen med et quadrokopter(QK) ble gjennomført av franskmannen Étienne Edmond Oehmichen i 18. februar 1920, med Oehmichen No. 2(Figur 1-1). Dette helikopteret bestod riktignok av flere rotorer enn 4, men prinsippet var det samme når det kom til løft, yaw, pitch og rull.



Figur 1-1 Oehmichen No.2 [1]

Figur 1-2, illustrerer hvordan propellene spinner for et X- oppsett. De grønne pilene indikerer at propellene spinner med klokka, CW. De røde pilene indikerer at propellene spinner mot klokka, CCW. Da propellene parvis spinner motsatt av hverandre, vil dette motvirke gyroeffektene som propellene genererer. De fleste quadrokoptere operer med faste vridninger på propellene. Det vil si at det kun er endringer på motor-turtall som kontrollerer farkosten. Et quadrokopter kan bevege seg i 6 frihetsgrader, rotere om 3-rotasjons akser, pitch, rull og yaw, og bevege seg langs 3-translasjons akser, høyre/venstre, frem/tilbake og opp/ned. For å stige i høyde, økes turtallet tilsvarende likt for alle motorene. For å bevege seg fremover(pitch), økes turtallet på motorparet bak i forhold til det fremre motorparet. For sidebevegelse(rull), økes turtallet på motorparet på motsatt side av ønsket bevegelse. For å åpnå endring i yaw, økes turtallet på motoreneparet som spinner samme vei, tilsvarende reduseres turtallet på det andre motorparet.

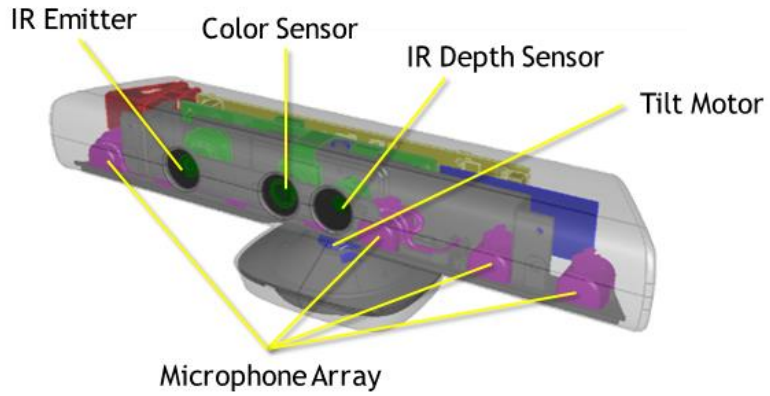


Figur 1-2 Propell rotasjon

Et quadrokopter er et dynamisk system, noe som gjør at den har behov for kontinuerlig regulering. For å regulere et quadrokopter må vinkelakselerasjonene måles. Til dette brukes et 3-akse gyroskop. Gyroskop måler kun vinkelendringer, det vil si at sensoren ikke kan detektere plane bevegelser. I tillegg vil gyrodrift sende et quadrokopter ut av kurs. For å bedre på dette kombineres gyroen gjerne med et akselerometer. Med et akselerometer vil det være mulig å måle hastighetsvariasjon i planet og i tillegg måle jordgravitasjon for å holde QK plan. Selv med et akselerometer vil et quadrokopter drifte ut av posisjon. Det er ikke før QK blir supplert med en GPS, at et QK kan bli fullstendig autonom. Men GPS vil derimot ikke fungere innendørs da signalenestyrken er for svake, -155dBW [2]. Det vil derfor være et behov for andre sensorer for å erstatte GPS innendørs. Prosjektet skal se på mulighetene ved å benyttes Kinect og «Motion Capture» for å erstatte denne.

1.3 Microsoft Kinect

Nyheten om Kinect, kodenavn *Project Natal*, ble annonsert 1. juni 2009 på spillmessen E3 og ble sluppet på markedet november 2010**.** Kinect er en bevegelsessensor med innebygde mikrofoner (Figur 1-3). Dette lar brukeren bruke gestikulasjoner og lyd, fremfor ordinære spill håndkontroller, til å samhandle med konsollen. Kinect ble utviklet for spillkonsollen Xbox 360, men det tok ikke lang tid før entusiaster, eksempelvis «OpenKinect» [3], fikk tilgang til datastrømmen. Også andre bransjer har også sett muligheter i Kinect. Tidlig 2012, lanserte Microsoft en Kinect med støtte til Windows.



Figur 1-3 – Kinect sensor spesifikasjoner

Mathworks Matlab fikk støtte for Kinect gjennom toolboxen, Image Acquisition Toolbox i versjon 2013b. Det er denne toolboxen som blir benyttet i dette prosjektet for å hente ut dybde data.

1.4 Struktur

Kapittel 2: Matematisk bakgrunn. Dette kapittelet inneholder beskrivelser av de matematiske formler og teoremer som ble benyttet til dette prosjektet.

Kapittel 3: Måling av posisjon og stilling med Kinect. Inneholder en beskrivelse av teknologien bak Microsofts Kinect sensor. Dette kapittelet inneholder kalibreringer av Kinect, deteksjon av quadrokopter i bilde og det ble funnet posisjon og orientering av quadrokopter med måleusikkerheter.

Kapittel 4: Autopilotdesign og uttesting. Inneholder skjematikken for kobling mellom en mikrokontroller til en fjernkontroll. Kommunikasjonen mellom PC og mikrokontroller. Til slutt en autopilot med måledata.

Kapittel 5: Konklusjon. Dette kapittelet inneholder konklusjon og videre arbeid.

Kapittel 6 Referanser og Appendiks: Inneholder de referanser som har blitt brukt under dette prosjektet. Appendiks inneholder de relevante Matlab-kodene og C-kode som har blitt brukt i dette prosjektet.

Figur 1-4 Oversikt over prosjektets grensesnitt

2 Matematisk bakgrunn

2.1 Rom og rammer

2.1.1 Referanserom og treghetsrom

Et referanserom er et fysisk rom. I navigasjonen er det her et fartøy fysisk befinner seg. For prosjektets del vil dette være den rammen quadrokopteret befinner seg i.

Et treghetsrom er et spesielt vektorrom hvor Newtons lover gjelder i sin enkleste form. I et treghetsrom er tiden homogen, det vil si klokker kan synkroniseres. Rommet er homogent og tidsuavhengig, dvs. at et fartøys posisjon i rommet ikke påvirker navigasjonen. Alle posisjoner i retninger, rommet og tiden er ekvivalent med hensyn på formulering av bevegelseslovene.

2.1.2 Vektorrom

Et vektorrom $\mathcal{V}$ er et sett som er lukket under en endelig vektor addisjon og vektor multiplikasjon. Et typisk eksempel er n-dimensjonal Euklidsk rom, ofte kalt Kartesisk rom, inneholdt i $\mathbb{R}^n$, hvor hvert element er representert i en liste bestående av n reelle numre, hvor skalarene er positive reelle, addisjonen er komponentvis og skalarmultiplikasjonen er multiplikasjon for hvert ledd, separat. [4]

For et generelt vektorrom er skalarene deler av et felt F, hvor $\mathcal{V}$ er kalt et vektorrom over F. Euklidsk n-rom, inneholdt $\mathbb{R}^n$, blir kalt et reelt vektorrom. For vektorrom inneholdt $\mathbb{C}^n$, blir kalt det komplekse vektorrom.

2.1.3 Affint rom

Affine rom er modellen av navigasjonsrammen, referanserammen. Det affine rom består av vektorer og punkter i rommet.

La $\mathcal{V}$ være vektorrommet over et område **K**, la A være ikke-tomt sett. Definerer i tillegg $p + a \in A$ for alle vektorer $a \in \mathcal{V}$ og punktet $p \in A$. Da gjelder:

1. $p + 0 = p$
2. $(p + a) + b = p + (a + b)$
3. For alle $q \in A$, eksisterer det en unik vektor $a \in \mathcal{V}$ slik at $q = p + a$

Her er $a, b \in \mathcal{V}$. Legg merke til at (1) er underforstått i (2) og (3). Her er da A det affine rom og **K** er koeffisient feltet.

I et affint rom er det mulig å fastsette et punkt og koordinataksene slik at for et hvert punkt i rommet kan representeres som en n-tuppel av dets koordinater. Hvert ordnet par av punktene A og B i det affine rom blir deretter assosiert med en vektor $\underline{AB}$. [5]

2.1.4 Koordinatsystem/ramme

I geometrien er et koordinatsystem en beskrivelse av et punkt i et vektorrom. For å kunne beskrive punktets posisjon i rommet, defineres et origo hvorav punktets posisjon måles ut ifra. I en referanse ramme kan det være flere koordinatsystem.

2.2 Indreprodukt

Indreprodukt brukes til blant annet til å beregne lengde av vektorer, finne ortogonalitet og for å projisere en vektor ned på en annen vektor.

For algebraiske vektorer. Har at:

$$\underline{x}, \underline{y} \in \mathbb{R}^n \qquad < \underline{x}, \underline{y} > = \underline{x}^T \underline{y} = x_1 y_2 + x_2 y_2 + .. + x_n y_n \qquad (1)$$

For geometriske vektorer:

$$< \vec{a}, \vec{b} > = \|\vec{a}\| \cdot \|\vec{b}\| \cdot cos\angle \vec{a}\vec{b} \qquad (2)$$

2.3 Normalisert krysskorrelasjon

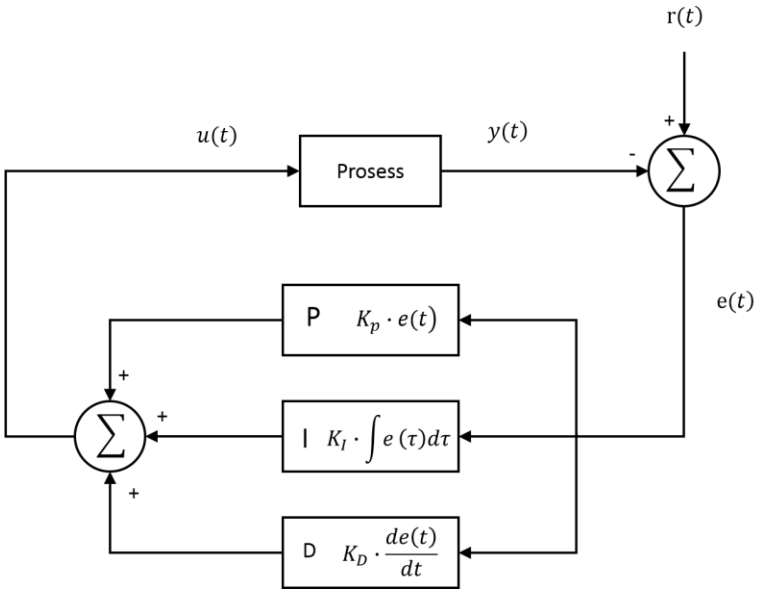
Det blir brukt normalisert krysskorrelasjon i dette prosjektet til bildebehandling. Normalisert krysskorrelasjon brukes for å finne aktuelle objekter i et bilde ved hjelp av et- eller flere referanse bilder. Referansebildene blir flyttet piksel for piksel i det aktuelle søkebilde. For hver posisjon blir den normaliserte krysskorrelasjons-koeffisienten beregnet med følgende formel:

$$\gamma(u,v) = \frac{\sum_{x,y}\left[f(x,y) - \overline{f}_{u,v}\right] \cdot \left[t(x-u, y-v) - \overline{t}\right]}{\sqrt{\sum_{x,y}\left[f(x,y) - \overline{f}_{u,v}\right]^2 \cdot \sum_{x,y}\left[t(x-u.y-v) - \overline{t}\right]^2}} \qquad (3)$$

Hvor f er bilde, $\overline{t}$ er gjennomsnittet av referansen, $\overline{f}_{u,v}$ er middelverdien av $f(x,y)$ i regionen under referansen.

2.4 PID-regulering

PID-regulatorer blir brukt i nær sagt alle prosesser i industrien, fra å regulere nivå i vanntank til regulering av pådrag til motorer. En PID-regulator er en tilbakekoblet kontroller, det vil si den måler avviket i en prosess, for deretter å korrigere for dette avviket. En PID-regulator består av tre deler, en Proporsjonal-, Integrator- og en Derivat-del. Den proporsjonale delen av regulatoren ser kun på avviket mellom settpunkt og målt verdi. Integral-leddet, summerer opp feilen over tid. Derivat-leddet ser på endringen av feilen, det vil si at derivat-leddet blir størst dersom endringen av feilen skjer raskt. Figur 2-1 er et blokkskjema for en standard PID-regulator.



Figur 2-1 PID-Regulator

For kontinuerlige systemer:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)d\tau + K_d \cdot \frac{d}{dt}e(t) \qquad (4)$$

For diskrete systemer:

$$u(k) = K_p \cdot e(k) + K_i \cdot \sum_{k_0}^{k} e(k) \cdot \Delta t + K_p \cdot \frac{e(k) - e(k-1)}{\Delta t} \qquad (5)$$

Hvor e er målt avvik fra settpunkt, u er pådraget og $K_p, K_i \; og \; K_d$ er konstanter som må tilpasses til hver prosess.

2.5 Elektronikk

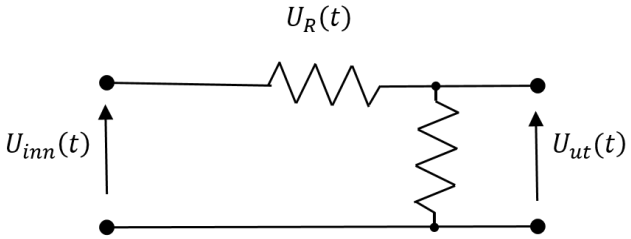
Underveis i prosjektet ble det klart at det var behov for elektronikk, hvorav følgende matematiske formler ble benyttet:

$$U(t) = R \cdot I(t) \tag{6}$$

Hvor $U(t)$ er spenningen, $I(t)$ er strømmen og R motstanden

2.5.1 Kirchhoff 's 2. lov

$$U_{inn}(t) = U_R(t) + U_{ut}(t) \tag{7}$$



2.5.2 Laplace transformasjon

En Laplace transformasjon er en integraltransformasjon. Denne transformasjonen brukes spesielt til å løse differensiallikninger. Den ensidige Laplace transformasjonen er definert som:

$$\mathcal{L}_t[f(t)](s) = \int_0^\infty f(t) \cdot e^{-st} dt \tag{8}$$

Hvor $f(t)$ er definert for $t \geq 0$

Tilsvarende for inverse transformasjon, er som følger:

$$\mathcal{L}^{-1}[F(s)] = \frac{1}{2\pi \cdot i} \lim_{T \to \infty} \int_0^{\gamma + i \cdot T} e^{st} F(s) ds \tag{9}$$

2.6 Koordinattransformasjon

2.6.1 Punkttransformasjon

To kameraer i et stereooppsett har hver sin ramme. Et punkt i rommet, p, observeres av begge kameraene. Generelt vil sammenhengen mellom kameraene være:

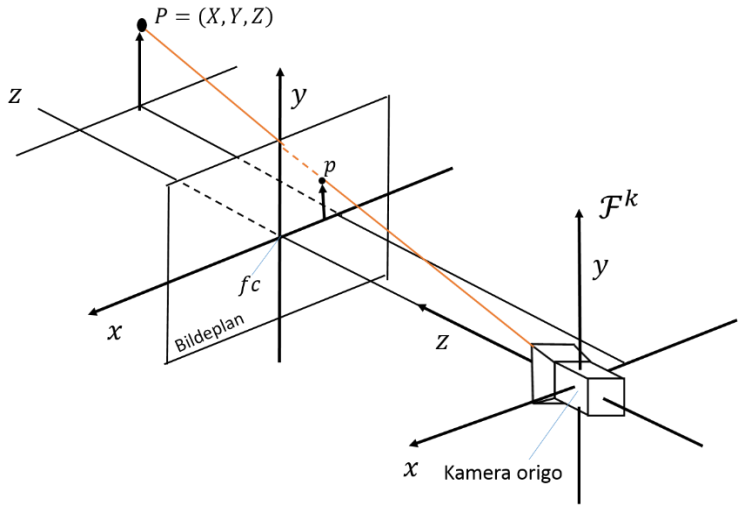
$$\underline{p}_1 = R_2^1 \cdot \underline{p}_2 + T_2^1 \qquad (10)$$

$$\underline{p} = 3 \times 1$$
$$T = 3 \times 1$$
$$R = 3 \times 3$$

Hvor T er translasjonen mellom rammene og R er rotasjonsmatrisen.

2.6.2 Bilde projeksjon

Figur 2-2 illustrerer hvordan et punkt i verden blir projektert på et 2D bildeplan.



Figur 2-2 Bildeprojektering

Et punkt i verden, P, vil bli projektert på et bildeplan, p, på følgende måte:

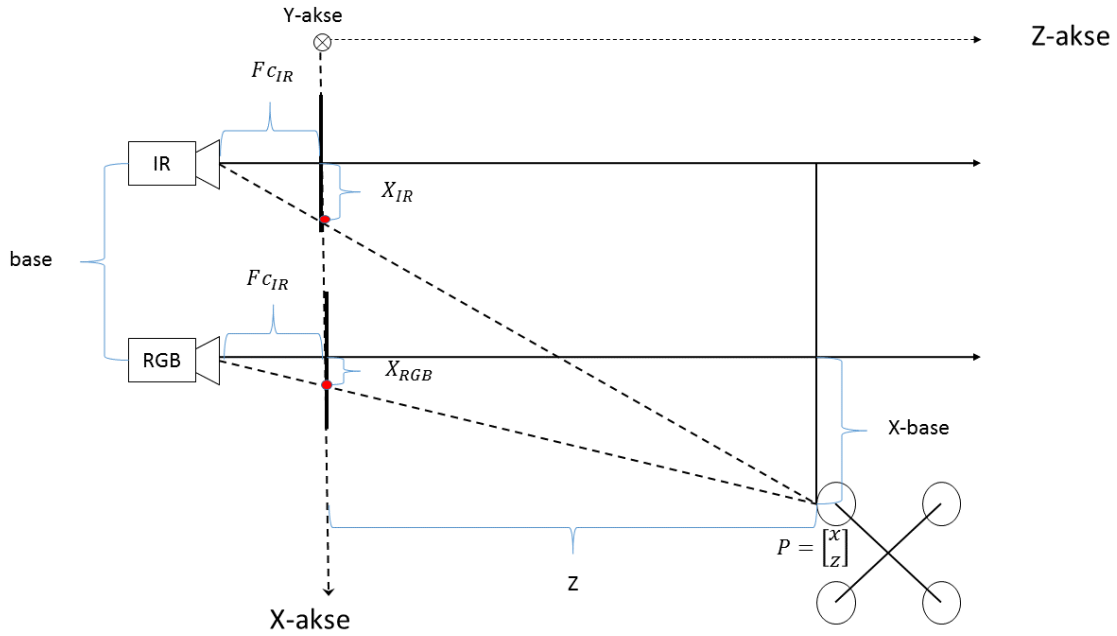
$$x = fc \cdot \frac{X}{Z} \qquad (11)$$

$$y = fc \cdot \frac{Y}{Z} \qquad (12)$$

Hvor fc er fokal avstanden fra linsa, det vil si avstanden til brennpunktet. Ser også av (8) og (9) at avstanden,Z, er vesentlig i forhold til projeksjonen av et punkt på et bildeplan.

2.6.3 Triangulering

Figur 2-3, viser hvordan et punkt i rommet, P, blir projektert på bildeplanet til stereokameraene.



Figur 2-3 Triangulering

$$z = \frac{X \cdot Fc_{ir}}{x_{IR}} \quad (13)$$

$$z = \frac{(X - base) \cdot Fc_{RGB}}{x_{RGB}} \quad (14)$$

Videre kan utledes:

$$z = \frac{Fc \cdot base}{x_{IR} - x_{RGB}} = \frac{Fc \cdot base}{d} \quad (15)$$

Ser at dybden, z, er invers proporsjonal med forholdet mellom x-komponentene,d. De resterende koordinatene i referanserammen blir som følger:

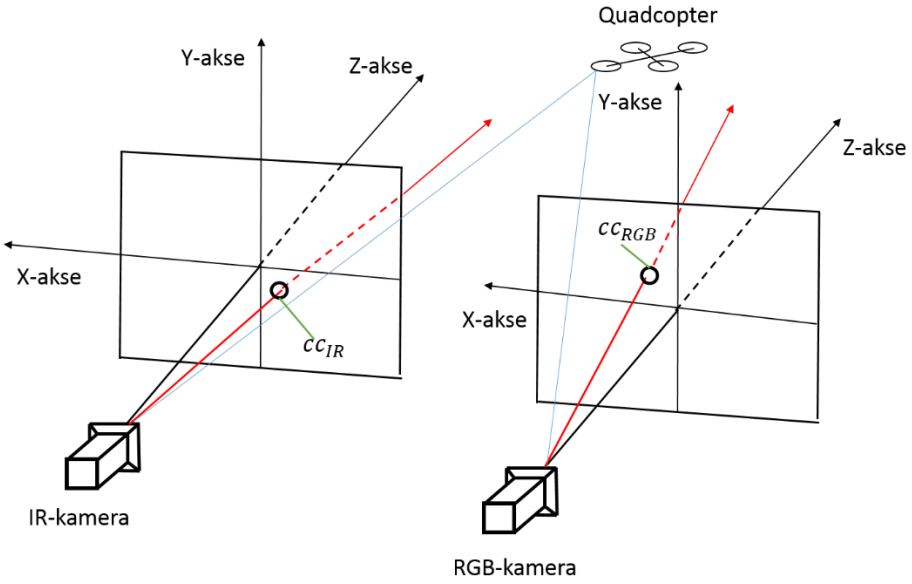
$$X = \frac{x_{IR} \cdot z}{Fc_{IR}}, \qquad X = base + \frac{x_{RGB} \cdot z}{Fc_{RGB}} \quad (16)$$

$$Y = \frac{y_{IR} \cdot z}{Fc_{IR}}, \qquad Y = \frac{y_{RGB} \cdot z}{Fc_{RGB}} \quad (17)$$

Hvor x_{IR} og y_{IR} er koordinatene til punktet, P, i IR-bilde. x_{RGB} og y_{RGB} er koordinatene for samme punkt for RGB-bilde. X,Y og z er koordinatene i rommet. Og Fc er fokallengden til linsene.

2.6.4 Ulikheter i linsene

For et stereo-oppsett vil ikke de prinsipielle midtpunktene i bildene, cc, eller fokal avstanden, Fc, være like mellom kameraene. Dette er konstanter som blir funnet ved kalibrering.



Kinect sensoren får dybde informasjonen direkte ut ifra dybde bilde. På den måten vil $z = dybde(x_{IR}, y_{IR})$. Transformasjonen for et punkt i dybdebilde,$p(x_{IR}, y_{IR})$, til fargebilde, blir som følger:

$$X = \frac{(x_{IR} - cc_{IR_x}) \cdot z}{Fc_{IR_x}} \quad (18)$$

$$Y = \frac{\left(y_{IR} - cc_{IR_y}\right) \cdot z}{Fc_{IR_y}} \quad (19)$$

$$z = p(x_{IR}, y_{IR}) \quad (20)$$

Hvor X,Y og z er koordinatene til P i rommet sett ifra IR-kamera.

P sett ifra RGB-kameraet blir som følger:

$$P_{RGB} = R \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + T \quad (21)$$

Hvor R og T er rotasjonsmatrisen og translasjonsmatrisen.

Koordinatene til punktet på RGB-bilde, blir som følger:

$$x_{RGB} = \frac{P_{RGB}(x) \cdot Fc_{RGB_x}}{P_{RGB_z}} + cc_{RGB_x} \quad (22)$$

$$y_{RGB} = \frac{P_{RGB}(y) \cdot Fc_{RGB_y}}{P_{RGB_z}} + cc_{RGB_y} \qquad (23)$$

3 Måling av posisjon og stilling med Kinect

Motion capture, gjerne kalt mocap, brukes for å finne posisjon, bevegelser til personer og/eller objekter ved hjelp av kamera. Det brukes gjerne flere kameraer fra flere vinkler. Gode systemer kan registrere små endringer som ansiktsuttrykk og andre små bevegelser. Dette er derfor en populær teknikk som flere filmskapere har benyttet seg flittig av. Filmregissøren Peter Jacksons tolkning av «Ringenes Herre», er kanskje en av de mest kjente eksemplene på bruken av motion capture.

For å kunne registrere bevegelse og posisjon/orientering (pose), blir markeringspunkter plassert på skuespillerens ledd og andre deler av kroppen. Det er ved hjelp av disse punktene at animasjoner kan gjengis via triangulering. Disse punktene kan være alt ifra lysdioder, da gjerne IR-lys, til reflekterende markeringsballer.

Selv om det kanskje er film folk flest forbinder med motion capture, har teknikken blitt brukt til militære applikasjoner, medisinske undersøkelser og annen forskning. Motion capture har allerede blitt brukt for regulering av quadrokoptere, og det er da særlig University of Pennsylvania [6] som har fått mye media oppmerksomhet for sine demonstrasjoner. Dette er store og dyre prosjekter med mange involverte.

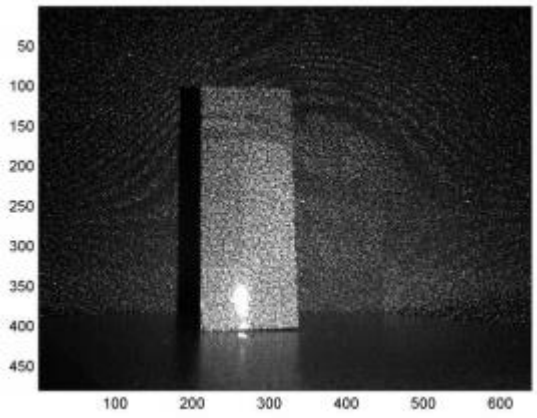
I nyere tid har motion capture teknologien blitt billigere og mer tilgjengelig for folk flest. Det er gjerne som tilbehør til spillkonsoller, og det er da spesielt Microsoft med sitt tilbehør til spillkonsollen Xbox360, Kinect, som leder an med denne teknologien.

3.1 Microsoft Kinect Sensor

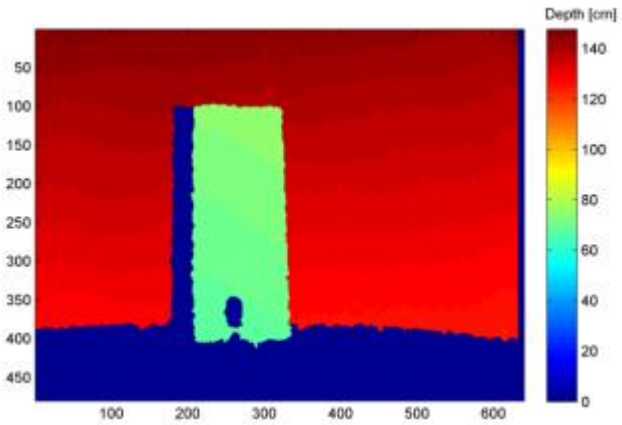
3.1.1 Dybde sensor

Kinect var en revolusjon innenfor rimelige motion capture systemer. Men det som kanskje gjør at Kinect skiller seg ut, er måten den «ser» et rom på da den ikke er avhengig av lysforhold. Kinect er i tillegg til RGB/farge-kamera utstyrt, med en infrarød(IR) laser-emitter og et IR-kamera. For å måle avstand til objekter benyttes triangulering mellom IR-emitteren og IR-kameraet. Ved å emittere et kjent speckel-mønster, vil størrelsen på dette mønsteret variere avhengig av avstanden. Dette mønsteret er lagret i Kinect sensoren med en referanse avstand, det vil si ved tilfeller der opplyste objekter endrer avstand til sensoren, vil dette speckelmønsteret endre seg. Speckel-mønsteret vil bli førskyvet i forhold referanse-mønsteret og det er denne forskyvningen som blir observert av IR-kameraet. Det er denne differansen mellom referanse punktet og det nye punktet som sier noe om avstanden til objektet. [7]

Figur 3-1, er et eksempel på hvordan IR-kameraet ser et objekt med et speckelmønster. Figur 3-2, tilsvarer dybdebilde basert på speckelmønsteret fra forrige figur. I de områder der speckelmønsteret ikke er synlig, er det heller ingen dybdedata.

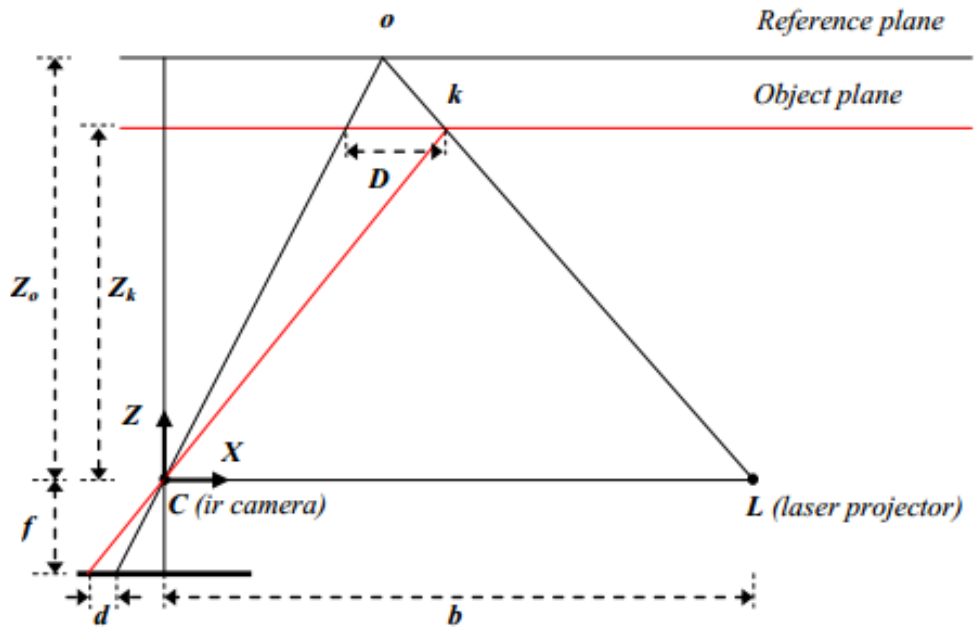


Figur 3-1 Infrarødt bilde med speckemønster



Figur 3-2 Det resulterende dybdebilde

Figur 3-3, illustrerer avstanden fra et punkt i rommet, k, relativt til Kinects referanseplan.



Figur 3-3 Kinect dybdemåling

I det punktet, k, flytter seg langs z-aksen vil speckelmønsteret forflytte seg. Punktet vil da forskyve seg på X-planet. Dette misforholdet vil måles på bilde, d.

Har at:

$$\frac{D}{d} = \frac{Z_0 - Z_k}{Z_0} \qquad (24)$$

Og:

$$\frac{d}{f} = \frac{D}{Z_k} \qquad (25)$$

Z_k er avstanden fra punktet, k, til Kinect planet. b er baselengden. f er fokallengden til IR-kamerat. D er forskyvningen av punktet, k, i rommet.

Løser ut D fra likning (25) og erstatter den i likning (24). Løser deretter for Z_k og får:

$$Z_k = \frac{Z_0}{1 + \frac{Z_0}{f \cdot b} \cdot d} \qquad (26)$$

I likning (26) er det kun d som endrer seg. Z_0 og f er konstanter. De resterende koordinatene i referanseramma blir som følger:

$$X_k = -\frac{Z_k}{f} \cdot (x_k - x_0 + \delta x) \quad (27)$$

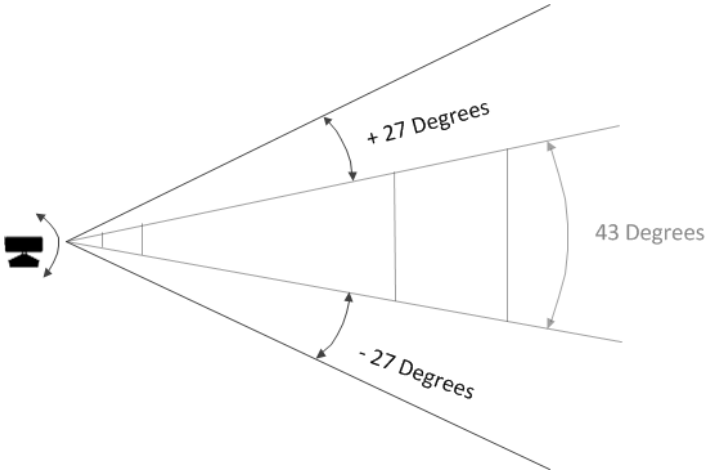
$$Y_k = -\frac{Z_k}{f} \cdot (y_k - y_0 + \delta y) \quad (28)$$

Her er x_k og y_k koordinatene i IR-bilde. Og δx og δy de prinsipielle punktene i bildene.

De konstante parameterne er verdier som blir funnet ved kalibrering, da dette er ikke offentlige tall. Denne trianguleringen er noe Kinect sensoren prosesserer internt og noe brukeren ikke trenger å ta hensyn til. Brukeren vil kun får ut dybde data i en 2D matrise/dybdebilde.

3.1.2 Andre spesifikasjoner

For uten IR-emitter og IR-kameraet, består Kinect sensoren av et RGB kamera som lagrer 3 kanals data i 1280x960 eller 640x480, oppløsning og et mikrofon-array bestående av 4 mikrofoner. Ved hjelp av disse mikrofonene kan lyd retnings bestemmes. I tillegg er Kinect utstyrt med akselerometer som kan benyttes til å estimere orientering til Kinect. Alle disse sensorene er pakket inn i et tilt-bart hus (Figur 3-4). [8]



Figur 3-4- Kinect vertikal tilt

Tabell 3-1 Øvrige spesifikasjoner

Kinect	Array Spesifikasjoner
Synsvinkel	43° vertikal og 57° horsontal
Vertikal tilt	$\pm 27°$
Bildefrekvens (RGB- og IR-kamera)	30 bilder per sekund
Lyd format	16-KHz, 24-bit mono puls kode modulasjon(PCM)
Lyd inngang karakteristikk	Fire-mikrofon array med 24-bit analog-til-digital konverter(ADC) og Kinect-resistent signal prosessering, inklusiv akustist ekko kansellering og støy kansellering
Akselerometer	En 2G/4G/8G akselerometer konfigurert for 2G området, med 1° øvre nøyaktighet

3.1.2.1 Dybdestrøm

Hvert bilde av dybdestrømmen er bygd opp av piksler som inneholder avstanden til objektet som er representert i pikselen. Avstanden måles i millimeter. Dybde bilde er tilgjengelig i 3 forskjellige oppløsninger: 640x480, 320x240 og 80x60. Bilde frekvensen er på 30 bilder/sekund for alle oppløsningene. Det er også 2 dybde moder tilgjengelig. For nærliggende scene (400-3000 millimeter) og standard scene (800-4000 millimeter).

3.1.2.2 Fargestrøm

Fargebilde strømmen er tilgjengelig i flere oppløsninger og formater. Formatet avgjør hvorvidt bildestrømmen blir kodet i RGB, YUV, eller Bayer. Det kan kun benyttes en oppløsning og en oppløsning av gangen.

Tabell 3-2 Bildedata

Farge format	Beskrivelse
RGB	32-bit, lineær X8R8G8B8-format farge bitmap i RGB farge. Ved oppløsning på 640x480, er bildefrekvensen 30 bilder/sekund. Ved 1280x960, er bildefrekvensen på 15 bilder/sekund
YUV	16-bit, gamma-korrigert lineær UYVY-formatert farge bitmap, hvor gamma korrigeringen i YUV rommet er ekvivalent til RGB gamma i RGB rommet. På grunn av at YUV strøm bruker 16 bit per pixel, bruker dette formatet mindre minne til å holde bitmap data og allokerer mindre buffer minne. Ved oppløsning på 640x480, er bildefrekvensen på 15 bilder/sekund. I dette formatet er det kun denne oppløsningen tilgjengelig
Bayer	32-bit, lineær X8R8G8B8-format farget bitmap, i RGB farge rom. Ved oppløsning på 1280X960, er bildefrekvensen på 12 bilder/sekund og for oppløsning på 640x480, er bildefrekvensen på 30 bilder/sekund.

3.2 Forberedelse

For å få tilgang til Kinect sensoren fra Matlab, var det nødvending å installere en support-tilleggspakke til Matlab Image Acquisition toolbox. Denne tilleggspakka ble hentet fra Mathworks sine hjemmesider [9].

Utgangspunktet for gjennomføring av dette prosjektet var å benytte RGB-bilde til å finne og registrere quadrokopteret, for deretter å gå inn i dybdedataene og finne avstandene til de tilsvarende punktene. Da Kinect består av to kameraer vil disse ha forskjellige linsespesifikasjoner og i tillegg til at de ikke står parallelt. For å finne hva et punkt i RGB-bilde tilsvarer i dybde-bilde, må denne transformasjonen bli beregnet via en kalibrering.

3.2.1 Kalibrering

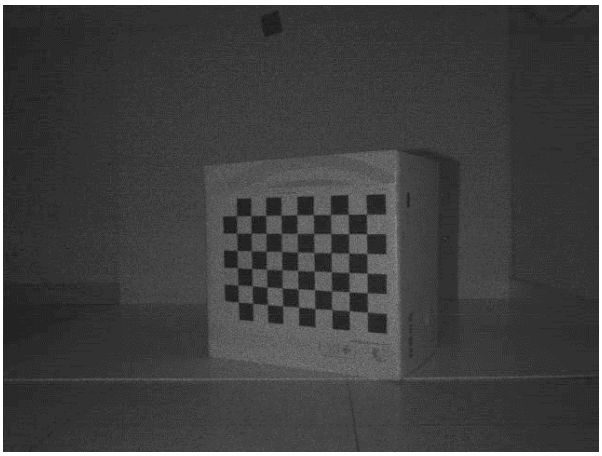
I prosjektet ble det gjort to typer kalibreringer. Den første kalibreringen ble gjort med en Matlab Toolbox. Den andre kalibreringen var mer en manuelt versjon.

Det er viktig å skille mellom hvilke bildedata som ble benyttet. For kalibreringen er det nødvendig at begge kameraene registrer de samme punktene på bildene. Da bildene fra dybde dataene kun sier noe om avstanden til objekter i rommet, vil ikke denne «se» mønstre som kalibreringen er avhengig av. For kalibreringen sin del, ble det benyttet IR-bilde fra IR-kameraet og fargebilde fra RGB-kameraet. For å ta bilde med IR-kameraet var det nødvendig å dekke til IR-emitteren med plast for å jevne ut speckel-mønsteret fra IR-emitteren.

3.2.1.1 Kalibrering med Matlab Toolbox

Da Kinect sensoren består av to kameraer, var det nødvendig å kalibrere begge disse. Til dette ble det benyttet en Matlab toolbox utviklet av Jean-Yves Bouguet [10]. Til denne kalibreringen ble det tatt en serie bilder av en eske med et sjakkmønsteret, hvorav rutene hadde mål på 25x25 mm. Det ble tatt 2 og 2 bilder med IR- og RGB-kameraet før esken ble flyttet til en annen posisjon med en annen orientering. Da data fra IR-kamera og RGB-kameraet blir sendt på samme kanal, er det ikke mulig å ta disse bildene samtidig. Det var derfor viktig at esken ikke ble forflyttet mellom opptakene fra IR- og RGB-kameraet.

Figur 3-5 og Figur 3-6 er et bildesettene som ble brukt under denne kalibreringen. Figurene viser tydelig forskjellene mellom linsene.



Figur 3-5 IR-kalibrerings bilde

Figur 3-6 RGB-kalibrerings bilde

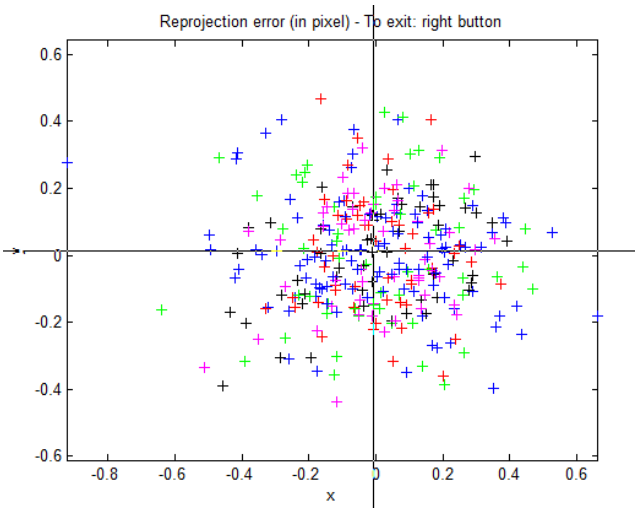
I første omgang ble hvert av kameraene kalibrert hver for seg [11]. Under denne prosessen går toolboxen igjennom bildene steg for steg, hvor brukeren må markere hoved-hjørnene på sjakkmønstret.

Fra denne kalibreringen fås følgende data av interesse for hvert av kameraene:

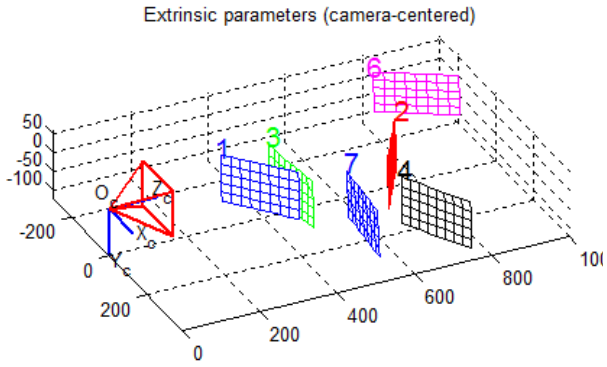
Fc-Fokal lengden:	Dette er avstanden til brennpunktet for linsa.
cc-Prinsipielle midtpunk :	Er det prinsipielle midtpunktet i bilde
alpha-:	Dette er skjevhet i linsa. Denne blir standard satt til null, da dette «kun» vil være relevant for vid-vinkel linser.

Dette er tall som blir forbedret i stereo-kalibreringen. Disse dataene ble derfor ikke lagt med.

Ser av Figur 3-7, hvor hver av «+»-ene sier noe om hvor mye programvaren misser hjørnene på sjakkmønstret. Figur 3-8, illustrerer de estimerte posisjonene til sjakkmønstret for hvert bilde.



Figur 3-7 Reprojisering feil



Figur 3-8 Estimert kamera posisjon

Ved stereo-kalibrering fås samme type data som kalibreringen til hvert av kamera, men da med forbedret usikkerhet enn det var på de individuelle kalibreringene. I tillegg til de forrige parameterne, blir det generert en rotasjonsmatrise og en translasjonsmatrise, R og T. Dataene for stereo-oppsettet ble som følger:

$$R = \begin{bmatrix} 0{,}9999 & -0{,}0042 & -0{,}0120 \\ 0{,}0042 & 1{,}0000 & -0{,}0052 \\ 0{,}0120 & 0{,}0052 & 0{,}9999 \end{bmatrix},\ T = \begin{bmatrix} 24.9135 \\ -0{,}2811 \\ -1{,}8194 \end{bmatrix}$$

$$Fc_{RGB} = \begin{bmatrix} 506{,}4384 \\ 504{,}5790 \end{bmatrix},\ Fc_{IR} = \begin{bmatrix} 573{,}2857 \\ 573{,}6506 \end{bmatrix},\ cc_{RGB} = \begin{bmatrix} 308{,}2479 \\ 228{,}4713 \end{bmatrix},\ cc_{IR} = \begin{bmatrix} 300{,}5625 \\ 239{,}0614 \end{bmatrix}$$

Rotasjonsmatrisen og translasjonsmatrisen er oppgitt i mm, hvor fokal-lengdene og de prinsipielle midtpunktene, er oppgitt i piksler.

Selv om kameralinsene på Kinect tilsynelatende kun er forskjøvet 25 mm horisontalt i forhold til hverandre, viser translasjonsmatrisen T, at fabrikkspesifikasjonene ikke stemmer. Det er tilsynelatende små endringer, men på lengre avstander vil dette gi en betydelig effekt på translasjonen mellom bildene.

Da dette prosjektet tar utgangspunkt i å finne objekter av interesse i RGB-bilde, for så å finne tilsvarende objekt i dybde-bilde, løses likning (18) til (23) for x_{ir} og y_{ir}

$$P_{rgb_x} = \frac{(x_{rgb} - cc_{rgb_x}) \cdot z}{fc_{rgb_x}}$$

$$P_{rgb_y} = \frac{\left(y_{rgb} - cc_{rgb_y}\right) \cdot z}{fc_{rgb_y}}$$

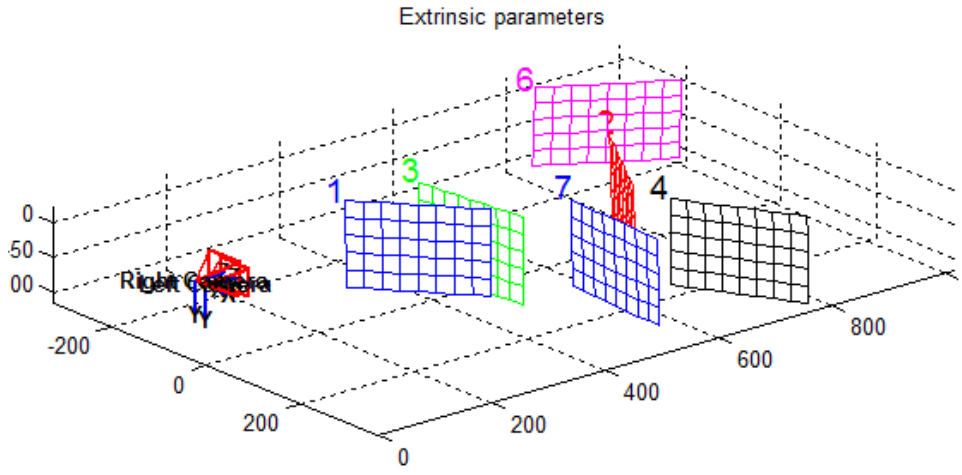
$$P_{ir} = R^T \cdot \begin{bmatrix} P_{rgb_x} \\ P_{rgb_y} \\ z \end{bmatrix} - T$$

$$x_{ir} = \frac{fc_{ir_x} \cdot P_{ir_x}}{z} + cc_{ir_x}$$

$$y_{ir} = \frac{fc_{ir_y} \cdot P_{ir_y}}{z} + cc_{ir_y}$$

Ser av formlene, ved et gitt punkt i RGB-bilde, så er dybden en vesentlig faktor for transformasjonen fra IR-bilde. Men denne avhengigheten blir vesentlig mindre ved økende distanser.

Figur 3-9 Estimert stereo kamera oppsett, viser hvor tett kameraene står i forhold til hverandre, samt hvor i rommet esken med sjakk mønster var under hvert opptak.



Figur 3-9 Estimert stereo kamera oppsett

3.2.1.2 Manuell kalibrering

Ettersom sammenhengen mellom pikslene på IR/dybde- og RGB-bilde er avhengige av avstanden til objektet, ble det gjort en mindre omfattende kalibrering for scener med lengre distanser til objekter av interesse.

Til denne kalibreringen ble det tatt to bilder av esken fra forrige delkapittel. Esken ble plassert i senter foran hver av kameralinsene. Det vil si at mellom hvert av bildene, ble esken flyttet asimut tilsvarende den avstanden som det ble målt til å være mellom linsene, dvs. 25 mm. Deretter ble bildene sammenliknet manuelt ved å legge de oppå hverandre. Det ble ikke tatt høyde for faktorer som skjevheter mellom kameraene seg imellom.

For å få to bilder som var «identiske» med hverandre, ble det bilde med størst åpningsvinkel, «klippet». Det vil si at hele rader i 2D-matrisa ble slettet, både de første og de siste radene. I dette tilfellet var det RGB-bilde som hadde størst åpningsvinkel. Etter at bilde var klippet, ble bilde utvidet til den samme oppløsning som bilde hadde i utgangspunktet, 640x480 piksler. Denne prosedyren ble gjentatt inntil esken på bildene hadde samme størrelse. Da RGB-bilde viste samme størrelse som på IR-bilde og hadde samme vertikal posisjon, måtte et av bildene bli forskjøvet horisontalt for at esken på bildene skulle overlappe. Det ble valgt at IR-bilde skulle forskyves tilsvarende de 25 mm.

På denne måten var det mulig å finne dybden til et piksel i RGB-bilde direkte utenom å bruke translasjons vektorer og rotasjons matrise.

Denne metoden tar ikke hensyn til avstanden til scenen, da det nettopp er avstanden som vil felle denne løsningen, i den grad avstanden varierer ved scener som er nærmere enn 1 meter. Denne kalibreringen fungerte derimot tilfredsstillende for scener som var dypere.

3.2.2 Fysisk størrelser på bilde

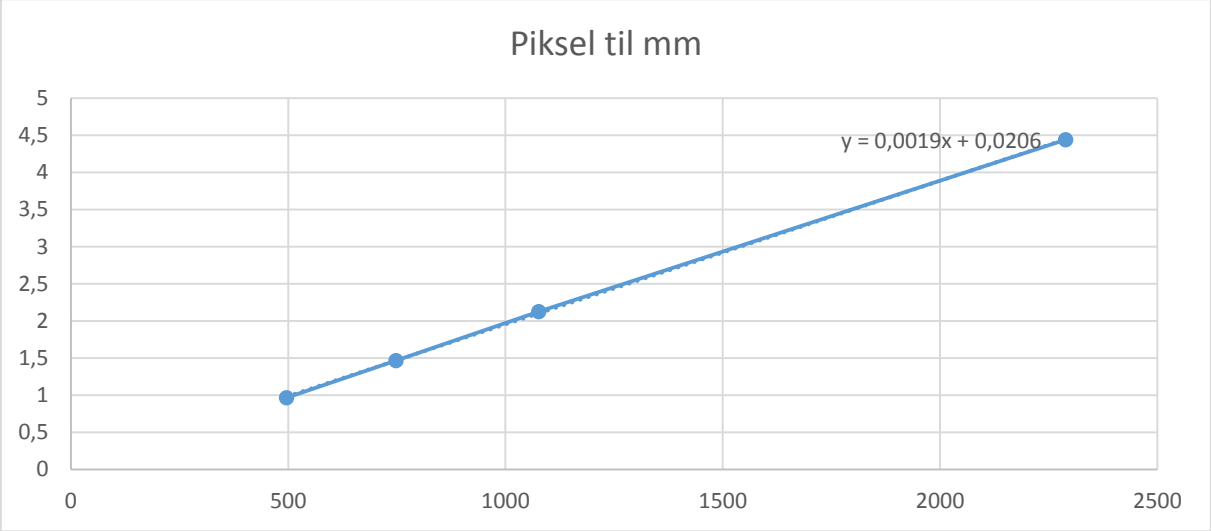
For å finne sammenhengen mellom dybde og fysiske mål på objekter på bilder, ble det tatt fire bilder av esken, fra forrige delkapittel, i økende avstand fra Kinect sensoren. For hver bilde ble avstanden registrert og antall piksler mellom hver av de sorte rutene, som vist i Figur 3-12 , talt.

Dataene ble som følger:

Tabell 3-3 Måledata for å finne fysisk størrelse

	Bilde 1	Bilde 2	Bilde 3	Bilde 4
Lengde i piksler	232,7	153,8	106	50,7
Dybde i mm	496	748	1076	2289
Fysisk lengde (mm)	225	225	225	255
Piksel/mm	0,967	1,463	2,123	4,438

Ser av Figur 3-10 at sammenhengen mellom dybde og piksler er tilnærmet lineær.



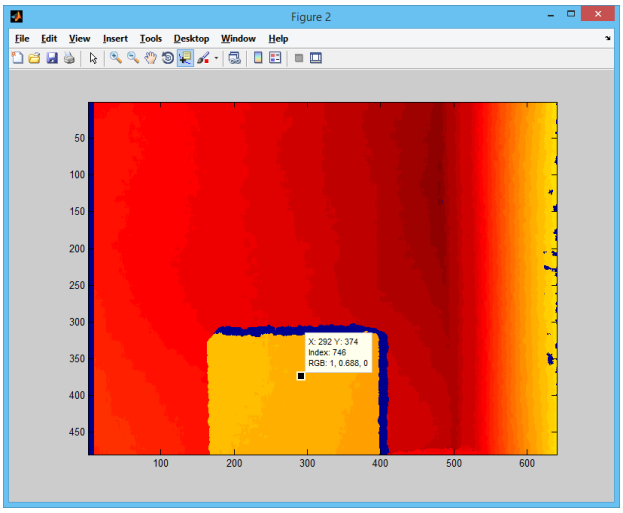
Figur 3-10 Piksel til mm

For å konvertere piksel størrelsen på et objekt til mm, ble denne formelen brukt.

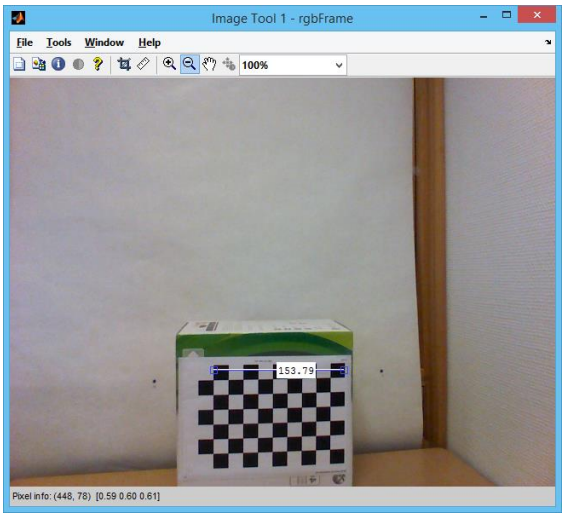
$$Lende(mm) = piksel * dybde(mm) * 1{,}9 * 10^{-3} + 2{,}027 * 10^{-2}$$

Siden nøyaktigheten til dybdesensoren er på $\pm$ 2 mm, settes det siste leddet til 0.

Eksempel på to bildene er på figurene nedenfor.



Figur 3-11 Dybde bilde



Figur 3-12 Pikselmål

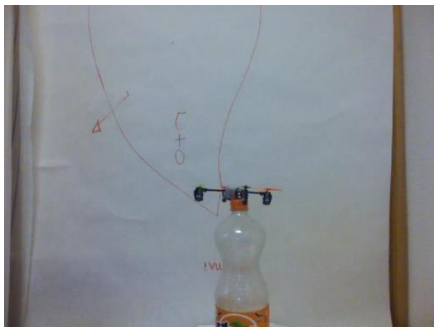
3.3 Deteksjon av quadrokopter

3.3.1 Endringsdeteksjon i bilde

Avhengig av hvor et quadrokopter opererer, vil bakgrunn og lyssettinger være faktorer som avgjør hvor godt det lar seg gjøre å finne og detektere quadrokopteret. Da en scene ikke vil variere i den grad et quadrokopter beveger seg mellom hvert av bildene, ble denne differansen undersøkt.

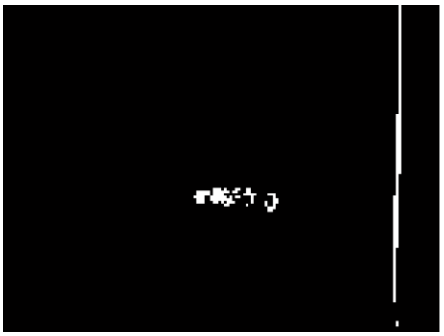
Det ble tatt to bilder med farge kameraet av en modell av et QK, hvor posisjonen ble endret mellom bildene. Det ble kun brukt RGB-kamera, så dybden ble ikke registrert. Hver av bildene bestod av 3 kanaler, det vil si at hver av bildene var 3D-matriser på 480x640x3. Matrisene ble først gjort om til to-dimensjonale matriser ved å vekte kanalene og addere disse sammen. Disse bildene ble da gråskala-bilder. Deretter ble differansen mellom bildene funnet. I de områdene hvor det ikke var noen endring mellom bildene, ble verdiene i bildematrisa satt til null. Tilsvarende ble områdene med endring satt til en verdi tilsvarende bevegelsen.

Figur 3-13, viser et av de to bildene som ble tatt. På Figur 3-14, er det tydelig at en quadrokopter-ramme skiller seg ut. Figur 3-15, er de svake differansene forsterket til et binært bilde.



Figur 3-13 Original bilde 1

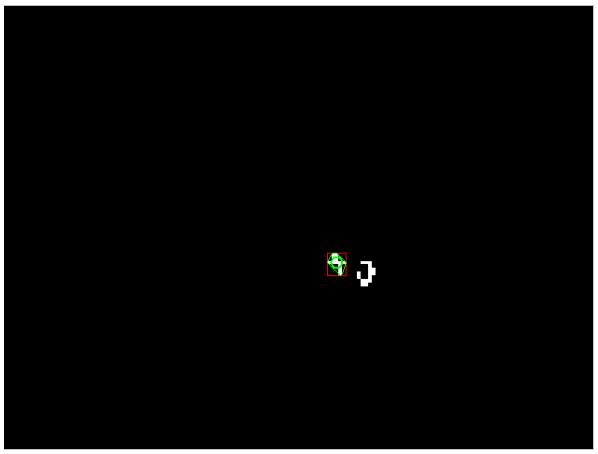
Figur 3-14 Skalert differanse bilde



Figur 3-15 Thresh metode

Fra Figur 3-15, er det betydelig støy på høyre side av bilde. For å løse dette, ble antall nabopiksler med ener-verdi talt. Deretter ble store sammenhengende ener-deler av bilde satt til null ved hjelp av «bwareaopen»-funksjonen i Matlab. Denne funksjonen ble kombinert med en xor-funksjon (eksklusiv eller), dermed ble for små og for store objekter filtrert ut.

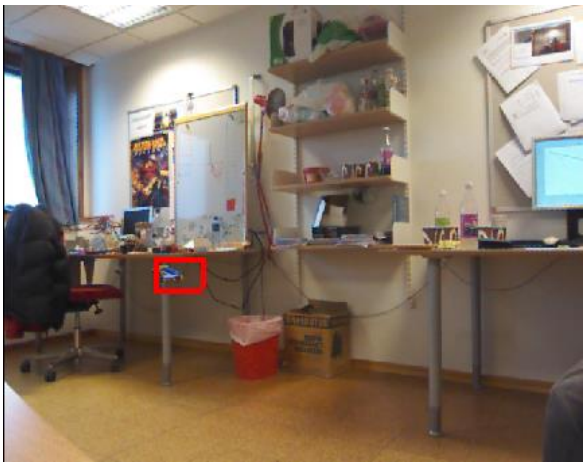
Det endelige resultatet ble som følger:



Figur 3-16 QK funnet

For å få koordinatene til de resterende objektene på bilde ble funksjonen «regionprops» benyttet. På figuren ovenfor er det den grønne sirkelen som markerer hvor funksjonen fant et objekt av interesse. I figuren er kun et objekt markert.

Denne metoden ble testet på «live» opptak. Det var riktignok ikke et QK som ble fløyet, men et mini helikopter. I figuren nedenfor viser hvordan helikopteret ble funnet og markert.



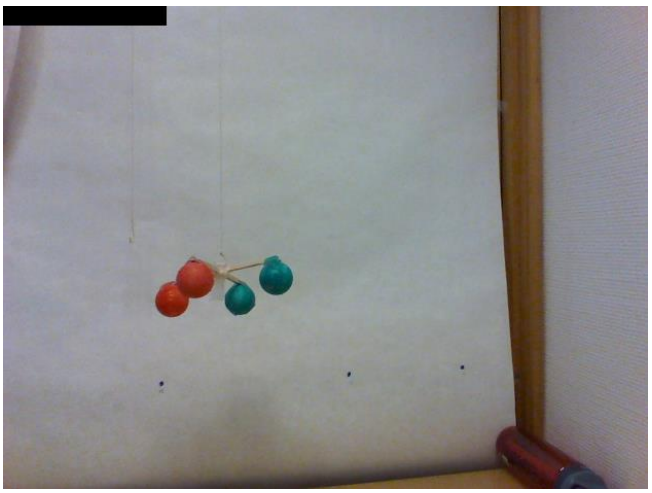
Figur 3-17 Endrings deteksjon

Dette denne metoden fungerte for å finne endringer i bilde, men metoden ble litt for følsom for forstyrrelser. For å dempe støyen ble det også forsøkt å skalere bildene til 1/4 del av den opprinnelige størrelsen. På den måten ble støyen midlet og differansebilde ble ikke så preget av støyen. Denne algoritmen klarte heller ikke å skille mellom et objekt av interesse, i dette tilfellet et helikopter, eller generell støy. For å finne QK måtte denne utstyres med kjente kjennetegn.

3.3.2 Finne kjennetegn på QK-ramme

For å finne QK, ble det valgt å lage scenen så «ren» som mulig. Til dette ble en hvit bakgrunn satt opp. Quadrokopter-ramma ble utstyrt med 4 fargede markeringsballer, et i hvert hjørne. To grønne representerte forsiden, og to røde for å representere baksiden. Fargevalget var heller ikke tilfeldig. Matlab's Image Acqusition toolbox lar brukeren sette hvilke data-format bilder fra kameraet skal være. Det er et alternativ å motta bildedata av type uint8 RGB format. Det vil si at et bilde er, valgt i dette tilfelle, en 480x640x3 matrise med verdier fra 0-255. Denne matrisen har da en kanal for hver av fargene, rød, grønn og blå. Det vil si at, eksempelvis røde objekter, får høye verdier i R-kanalen og visa versa for de resterende kanalene.

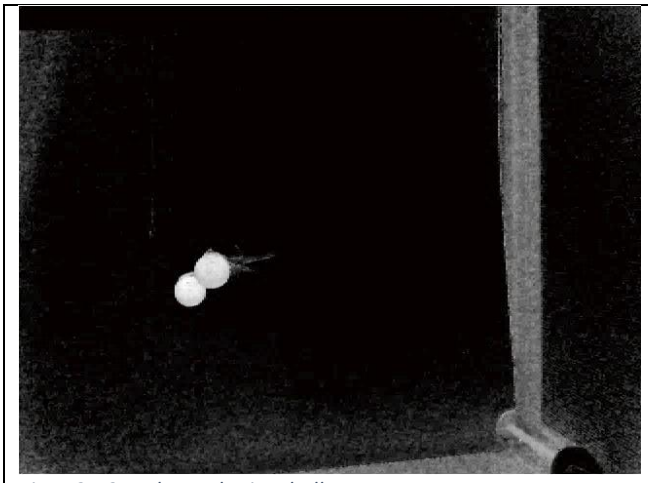
Figur 3-18, er det et bilde tatt av QK-ramma med markeringsballene.



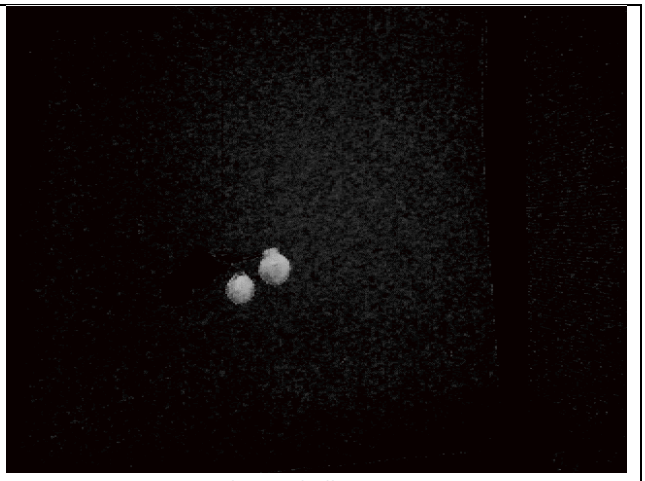
Figur 3-18 Test-ramme

Valget av oppløsning på bilde var grunnet størrelsen på dataene som måtte prosesseres. Denne oppløsningen er også maksimal oppløsning på dybdedataene og den oppløsningen kalibreringene utført med. I tillegg vil scener som Kinect operer i innendørs, sjelden bli større enn 4-5 meter, noe som gjør at oppløsningen vil dekke markeringsballene som ble brukt.

For å finne markeringsballene på bilde, ble det laget en kopi av originalbilde som deretter ble vektet til et gråskala bilde, på samme måte som i 3.3.1. Deretter ble originalbildet røde og grønne kanaler subtrahert med gråskalabilde hver for seg. På den måten ble de to resulterende bildene svart/hvit bilder hvor de røde og grønne fargene fremsto som lyse objekter. I figurene nedenfor er det tydelig hvor godt Matlab klarer å skille mellom fargene.



Figur 3-19 Røde markeringsballer



Figur 3-20 Grønne markeringsballer

Etter at rød- og grønn-kanal ble trukket ut, ble de lyse elementene forsterket og filtrert på samme vis som i forrige delkapittel.

Med to bilder som hver seg inneholdt røde og grønne markeringsballer, ble det benyttet Matlabs «Blob Analysis» for å finne posisjonen til markeringsballene. Denne funksjonen søker i de binære bilde-matrisene og kan finne «lyse» objekter, måler størrelsen og finner sentroiden. Det var disse målene ble brukt for å sjekke om det faktisk var markeringsballene som blir observert.

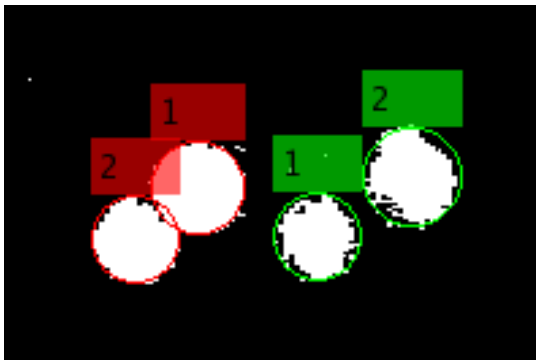
«Blob Analysis» -metoden er rask og effektiv, 0,001099 sekunder for prosjektets PC (Vil avhenge av PC kraft). Et problem som er tydelig i figuren ovenfor, er i de tilfeller hvor markeringsballer med samme farge overlapper hverandre. For dette forsøket, tolket «Blob Analysis» -funksjonen de røde markeringsballene som en- og vesentlig større ball. Dette var et problem som først og fremst hadde med oppsett å gjøre da QK-ramma var i samme plan som Kinect sensoren.

For å løse dette problemet, ble det lett etter kontrastdifferansene som skilte de to markeringsballene. I Matlab Image Processing Toolbox er det flere funksjoner som gjør det mulig å transformere bilde slik at markeringsballene kan skille seg mer ut og for deretter å separeres. Dette er en tung prosessering og tar lang tid. Avhengig av hvilke funksjon som blir brukt, kan dette ta opp til 0,226 sekunder å prosessere et enkelt bilde. Å benytte denne metoden til samtid-deteksjon av markeringsballene, vil fungere dårlig.

I dette forsøket var ikke tiden kritisk, hvor det da ble benyttet en annen funksjon, «imfindcircle». Denne funksjonen benytter en *Circular Hough* transformasjon for å finne sirkler i et bilde.

I figurene under klarer Matlab å separer og identifisere ballene.

Figur 3-21 QK-testramme



Figur 3-22 Separerte markeringsballer

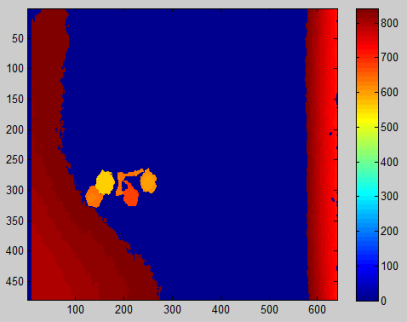
Resultatet ble at markeringsballene ble separert og størrelsen og koordinatene ble funnet.

3.3.2.1 Normalisert kryss-korrelasjon

Som nevnt i delkapittel 3.2.1, er avstanden til interessentene i bildene avgjørende for forskyvningen mellom dybdebilde og fargebilde. For dette oppsettet var scenen under 1 meter i dybde. Det vil si at forskyvningen blir veldig stor mellom hvert av bildene. For å finne denne forskyvningen ble det benyttet normalisert-krysskorrelasjon.

Som beskrevet i delkapittel 2.3, er normalisert-krysskorrelasjon avhengig av et referansebilde. Referansebilde ble derfor klippet ut ifra bilde på Figur 3-22, basert på koordinatene fra «Blob Analysis» -funksjonen, som deretter ble brukt i søket etter et liknende objekt i dybdebilde. Da offseten var kjent var det mulig å gå direkte inn i dybde-matrisen og lese av avstanden til de ønskede markeringsballene.

Figur 3-23 er dybdebilde av testramma. Fargeskalaen går fra 0-850 mm, det vil si avstanden fra objektene til Kinect XY-plan. Piksler som ikke har noen dybdeinformasjon blir satt til null. Figur 3-24 er referanse bilde som ble brukt til den normaliserte-krysskorrelasjonen. Figur 3-25 viser fargebilde og dybdebilde lagt over hverandre, hvor det er tydelig hvor stor forskyvning det er mellom bildene.



Figur 3-23 Dybde bilde av testramme

Figur 3-24 Referanse bilde

Figur 3-25 Forskyvning mellom RGB og dybdeFigur 3-25

I dette forsøket, ble det funnet en offset med en korrelasjonsverdi på 0,63. Denne offseten viste seg å treffe tilstrekkelig til at det var mulig å gå direkte inn i dybdebilde med offset-koordinatene, og få dybden til hvert av markeringsballene.

Det ble gjort et forsøk på å søke etter markeringsballer i dybdebilde fremfor å lete i fargebilde først. Dette ble gjort ved å steppe igjennom hele dybdespekteret og lete etter sirkulære objekter for hvert stepp. Dette viste seg å ikke fungere. Denne metoden tok i tillegg veldig lang tid å gjennomføre for hvert bilde.

3.3.3 Måling av markeringsballer

For å bekrefte størrelsen til markeringsballene, var det av interesse å finne ut hvor godt Matlab kunne estimere størrelsene på markeringsballene på bilde. Til dette ble foregående oppsett og algoritmer benyttet. I tillegg til auto-måling ble det også gjort en manuell måling direkte på bilde.

Ballene som benyttes som markeringsballer, er ordinære bordtennisballer som har blitt farget. Disse kulene har en radius på 2 cm. I testbildet ble den grønne markeringsballen lengst til høyre målt. I tillegg til automatisk måling, ble den sammen ballen målt manuelt i bilde.

Dataene ble som følger:

Tabell 3-4 Måling av markeringsball

Målinger	Radius i piksler(Rp)	Dybde i mm(Dm)	Radius(mm)=$Rp * Dm * 1{,}933 * 10^{-3}$	Differanse
Manuel måling	16,8 piksler	600 mm	19.5 mm	-0,5 mm
Auto-måling	17 piksler	601 mm	21 mm	1 mm

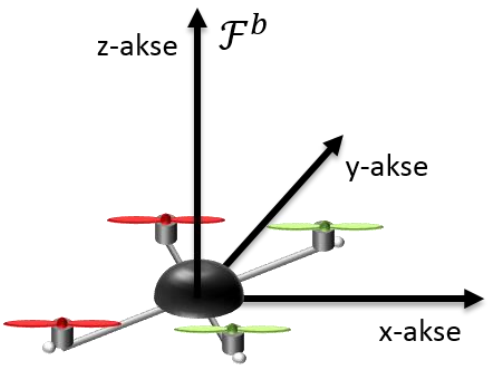
Målingene viser at feilen er på ca. 1mm. Da målinger fra forsøk veksler mellom $\pm$2mm. Anses dette resultatet som meget bra.

3.4 Stilling på Quadrokopter

I dette delkapittelet blir rammene i systemet definert og retningskosinmatriser og translasjonsvektorer mellom rammen funnet. Det bli også sett på støy på posisjonsestimeringen. Heretter ble det benyttet kalibreringsdata, fremfor normalisert-krysskorrelasjon for å finne offseten mellom bildene.

3.4.1 Quadrokopter-rammen

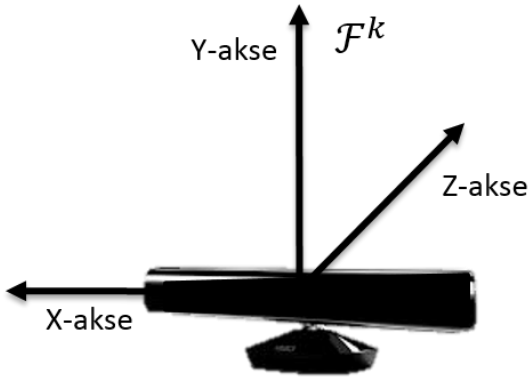
Rammen til quadrokopteret ligger fast i massesenteret. Hvor positiv x-retning er fremover(pitch). Positiv y-retning er til venstre for kjøreretningen(rull). Positiv z-retning står vinkelrett på xy-planet, oppover for quadrocopteret. Denne rammen(Figur 3-26) blir kalt for basis rammen, $\mathcal{F}^b$



Figur 3-26 Quadrokopter ramme

3.4.2 Kinect-rammen

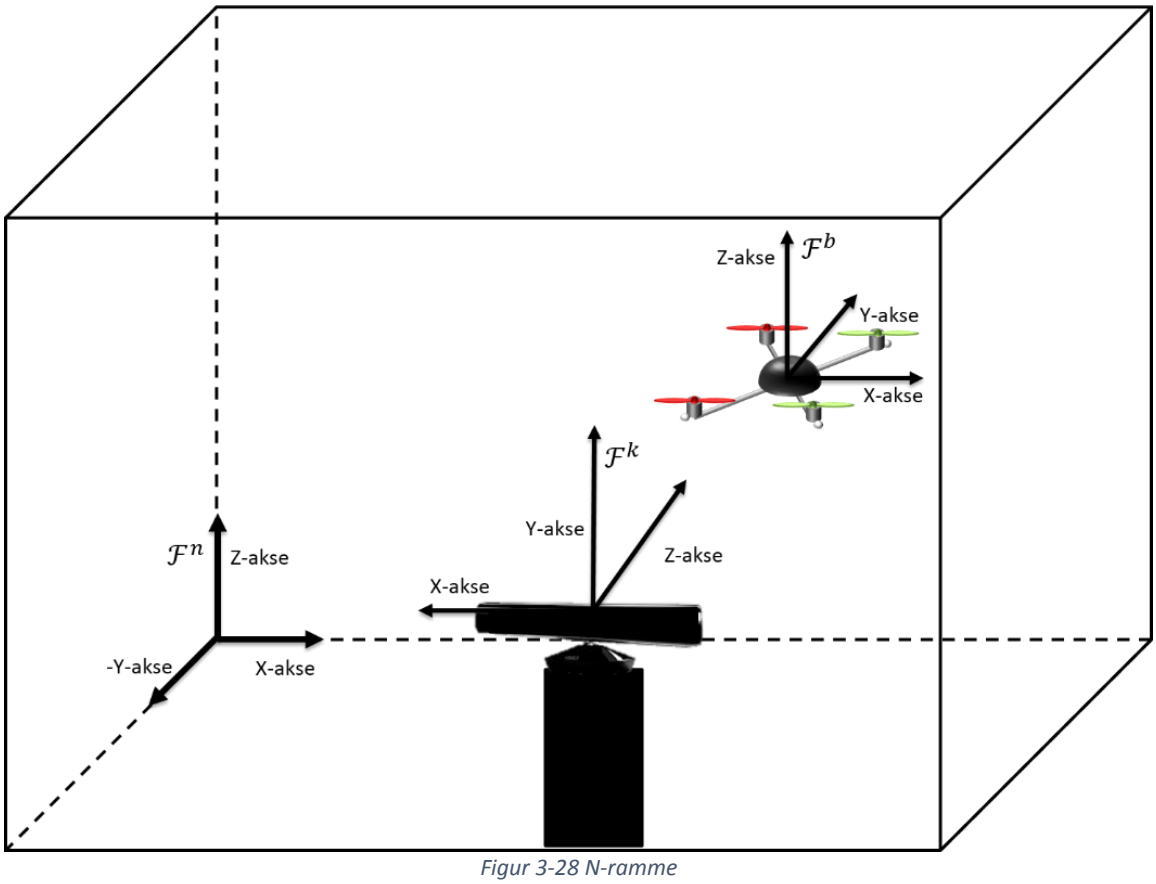
Kinect-rammen (Figur 3-27), er den rammen som observerer navigasjons-rammen og quadrokopter-rammen. I dette prosjektet defineres x-retning til venstre for Kinect sensoren, sett bak ifra. Positiv y-retning er opp for sensoren og z-retningen står vinkelrett på xy-planet ut ifra Kinect sensoren.



Figur 3-27 Kinect ramme

3.4.3 Navigasjonsrammen

Navigasjons-rammen (Figur 3-28) er den rammen som quadrokopteret skal operere i. I figuren nedenfor illustrerer hvordan n-ramma er definert i forhold til de andre rammene.



Figur 3-28 N-ramme

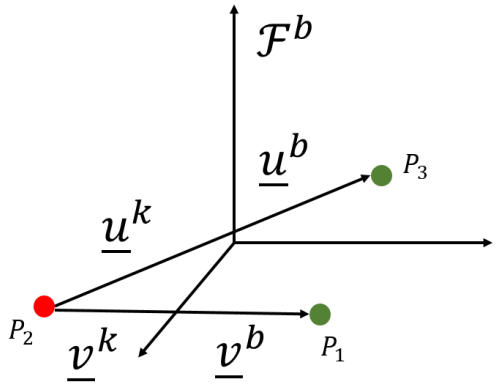
3.4.4 Quadrokopteret i Kinect-rammen

For å finne stilling og posisjonen til QK i rommet, er det kun nødvendig med 3 makeringsballer synlig og målbare for Kinect. Det er kun disse ballene som vil bli overført til Kinect rammen, $\mathcal{F}^k$. Det som er kjent er, markeringsballene på quadrokopteret i forhold til quadrocopter rammen, $\mathcal{F}^b$, og de observerte vektorene til ballene fra Kinect rammen.

Hvor $\underline{P_1^b}$, $\underline{P_2^b}$ og $\underline{P_3^b}$ er vektorene ut til markeringsballene sett ifra quadrokopter-rammen. For å finne disse vektorene ble punktene valgt til å være: P_1, den grønne ballen lengst til høyre i Figur 3-30, den røde markeringsballen som er fremst i bilde valgt til å være P_2 og til slutt hvor den siste grønne ballen ble P_3. Vektorene er som følger:

$$\underline{P_1^b} = \begin{pmatrix} 50 \\ -50 \\ 0 \end{pmatrix}, \underline{P_2^b} = \begin{pmatrix} -50 \\ -50 \\ 0 \end{pmatrix}, \underline{P_3^b} = \begin{pmatrix} 50 \\ 50 \\ 0 \end{pmatrix}$$

Figur 3-29, illustrerer hvilke markeringsballer, i Figur 3-30, som blir definert til hver av punktene.



Figur 3-29 Vektorer til markerinspunktene

Figur 3-30 QK-testramme

Basisvektoren til quadrokopteret, sett ifra i Kinect-rammen er som følger:

$$\underline{P_b^k} = \quad \underline{P_i^k} - R_b^k \underline{P_i^b} \quad i \in (1,3) \qquad (29)$$

Da det vektorene $\underline{P_1^k}, \underline{P_2^k}, \underline{P_3^k}$ måles og vektorene: $\underline{P_1^b}, \underline{P_2^b}, \underline{P_3^b}$ er kjent og fast fra quadrokopter rammen, kan retningskosinmatrisa bli funnet på følgende måte:

$$\underline{u}^k = \underline{P_2^k} - \underline{P_1^k} \quad (30)$$

$$\underline{v}^k = \underline{P_3^k} - \underline{P_1^k} \quad (31)$$

$$\underline{u}^b = \underline{P_2^b} - \underline{P_1^b} \quad (32)$$

$$\underline{v}^b = \underline{P_3^b} - \underline{P_1^b} \quad (33)$$

1. Definere først to stillingsmatriser R_c^b og R_c^n

a.

$$\underline{c_1^b} = \frac{\underline{u}^b}{\left\|\underline{u}^b\right\|} \quad (34)$$

$$\underline{c_2^b} = \frac{\underline{u}^b \times \underline{v}^b}{\left\|\underline{u}^b \times \underline{v}^b\right\|} \quad (35)$$

$$\underline{c_3^b} = \underline{c_1^b} \times \underline{c_2^b} \quad (36)$$

$$R_c^b = [\underline{c_1^b}\ \underline{b_2^b}\ \underline{c_3^b}] \quad (37)$$

b.

$$\underline{c_1^k} = \frac{\underline{u}^k}{\left\|\underline{u}^k\right\|} \quad (38)$$

$$\underline{c_2^k} = \frac{\underline{u}^k \times \underline{v}^k}{\left\|\underline{u}^k \times \underline{v}^k\right\|} \quad (39)$$

$$\underline{c_3^k} = \underline{c_1^k} \times \underline{c_2^k} \quad (40)$$

$$R_c^k = [\underline{c_1^k}\ \underline{c_2^k}\ \underline{c_3^k}] \quad (41)$$

2. Ved løse likninga mhp. R_b^k fås

$$R_c^k = R_b^k R_c^b \quad (42)$$

$$R_b^k = R_c^k (R_c^b)^T \quad (43)$$

Vektorene ble funnet både via Matlab-algoritmene og ved manuell måling på bildene. Dataene ble som følger:

Fra likning (32) og (33) fås :

$$\underline{u}^b = \begin{pmatrix} -100 \\ 0 \\ 0 \end{pmatrix}, \underline{v}^n = \begin{pmatrix} 0 \\ 100 \\ 0 \end{pmatrix}$$

Ved likning (34), (35) og (36) ble:

$$\underline{c}_1^b = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}, \underline{c}_2^b = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}, \underline{c}_3^b = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$$

Ved likning (37) fås.

$$R_c^b = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

$\underline{P}_1^k, \underline{P}_2^k \; og \; \underline{P}_3^k$ ved manuell måling:

$$\underline{P}_1^k = \begin{pmatrix} 42 \cdot 600 \cdot 0{,}001933 \\ -(30 \cdot 600 \cdot 0{,}001933) \\ 600 \end{pmatrix} = \begin{pmatrix} 49 \\ -35 \\ 600 \end{pmatrix}$$

$$\underline{P}_2^k = \begin{pmatrix} 122 \cdot 555 \cdot 0{,}001933 \\ -(122 \cdot 555 \cdot 0{,}001933) \\ 555 \end{pmatrix} = \begin{pmatrix} 131 \\ -57 \\ 555 \end{pmatrix}$$

$$\underline{P}_3^k = \begin{pmatrix} 76 \cdot 674 \cdot 0{,}001933 \\ -(50 \cdot 674 \cdot 0{,}001933) \\ 674 \end{pmatrix} = \begin{pmatrix} 99 \\ -65 \\ 674 \end{pmatrix}$$

Ved likning (30) og (31) ble.

$$\underline{u}^k = \begin{pmatrix} 82 \\ -22 \\ -45 \end{pmatrix}, \quad \underline{v}^k = \begin{pmatrix} 0 \\ 10 \\ 0 \end{pmatrix}$$

Ved likning (38), (39) og (40) ble.

$$\underline{c}_1^k = \begin{pmatrix} 0{,}8537 \\ -0{,}2292 \\ -0{,}4675 \end{pmatrix}, \underline{c}_2^k = \begin{pmatrix} -24{,}9849 \\ -83{,}4441 \\ -13{,}8378 \end{pmatrix}, \underline{c}_3^k = \begin{pmatrix} -35{,}8605 \\ 25{,}8327 \\ -78{,}1129 \end{pmatrix}$$

$$R_c^k = \begin{pmatrix} 0{,}8537 & -29{,}9849 & -35{,}8405 \\ -0{,}2292 & -83{,}441 & 25{,}8327 \\ -0{,}4675 & -13{,}8378 & -78{,}1129 \end{pmatrix}$$

Ved linking (43) ble den endelige retningkosinmatrisa.

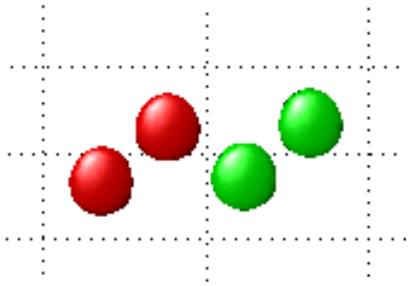
$$R_b^k = \begin{pmatrix} -0{,}8770 & 0{,}4050 & 0{,}1252 \\ 0{,}0164 & -0{,}2856 & 0{,}8344 \\ 0{,}4803 & 0{,}7297 & 0{,}2571 \end{pmatrix}$$

For å finne $\underline{P_b^k}$, kan $\underline{P_1^k}, \underline{P_2^k}\ og\ \underline{P_3^k}$ benyttes, men dette viste seg til å bli forskjellige resultater, nedenfor er gjennomsnittverdien med medfølgende standardavvik.

$$\underline{P_b^k} = \begin{pmatrix} 114 \\ -49 \\ 614 \end{pmatrix} \text{ , standardavvik: } \begin{pmatrix} 7{,}7 \\ 1 \\ 1{,}5 \end{pmatrix}$$

Ved å la Matlab søke igjennom bilde ble retningskosinmatrisa, R_b^n, og basisvektoren, $\underline{P_b^n}$,som følger:

$$R_b^k = \begin{pmatrix} -0{,}8736 & 0{,}4064 & 0{,}1656 \\ 0{,}0200 & -0{,}3101 & 0{,}8467 \\ 0{,}4861 & 0{,}7430 & 0{,}2628 \end{pmatrix}, \underline{P_b^k} = \begin{pmatrix} 115 \\ -50 \\ 616 \end{pmatrix} \text{ , standardavvik: } \begin{pmatrix} 8{,}8 \\ 1{,}5 \\ 1{,}8 \end{pmatrix}$$



Figur 3-31 QK posisjonsplott

I Figur 3-31 er det tydelig at Matlab gjør en god jobb med å finne og plotte quadrokopteret i koordinatsystemet til Kinect sensoren, $\mathcal{F}^k$. Standardavviket til basisvektoren var noe dårligere da Matlab søkte igjennom bilde, enn ved den manuelle målingen. Men dette var usikkerhet på opp mot en 1 mm, som anses som godt.

3.4.5 Posisjon til quadropkopter i n-rammen

Med erfaring fra punkt 3.4.1, var det klart at Kinect sin posisjon i forhold til quadrokopteret og navigasjons ramma var avgjørende for hvor godt det lot seg gjøre å registrere alle markeringsballene. Kinect sensoren ble derfor plassert ovenfor navigasjonsramma slik at hvert av markeringsballene var synlige til enhver tid. Det ble også valgt å benytte kun av 3 markeringsballer med henholdsvis 3 forskjellige farger. Disse ble da valgt til å være rød, grønn og blå. Grønn ball ble valgt til å representere fremsiden av QK, P1, blå ball representerte venstre side for kjøreretningen, P2, og rød ball representerte høyre side av kjøreretningen, P3.

For å gjøre oppsettet fleksibelt ble det valgt montere symboler på bakgrunnen av navigasjonsramma for å representere aksene i rammen. Navigasjonsrammen ble definert hver oppstart ved å lete etter disse symbolene og finne retningkosinmatrisa og basisvektoren fra n-ramma til kinect-ramma.

Disse symbolene ble påklistret direkte på bakgrunnen langs aksene, 300 mm fra basisen. Tegnene ble valgt til å være et «addisjons-tegn» for å representere $q1$, likhetstegn for å representere $q2$ og en trekant for å representere $q3$.

Tegnene er illustrert i figurene nedenfor:

Figur 3-32 q1

Figur 3-33 q2

Figur 3-34 q3

For å finne posisjonen til QK i navigasjonsramma, ble det fløyet et quadrokopter, manuelt, innenfor syn rekkevidden til Kinect sensoren. Under denne flyvningen det det gjort opptak av RGB-kameraet og dybdeopptak. Opptaket ble tatt med ca.30 bilder per sekund, og ble gjort over 20 sekunder.

Etter opptaket ble symbolene, som markerer navigasjons rammen, funnet ved hjelp av normalisert krysskorrelasjon. Hvor det var figurene ovenfor som ble brukt som referansebilder.

Vektorene til punktene som markerte aksene i navigasjonsrammen ble da som følger:

$$\underline{q_1^n} = \begin{bmatrix} 300 \\ 0 \\ 0 \end{bmatrix}, \underline{q_2^n} = \begin{bmatrix} 0 \\ 300 \\ 0 \end{bmatrix}, \underline{q_3^n} = \begin{bmatrix} 0 \\ 0 \\ 300 \end{bmatrix} \quad (44)$$

$$\underline{q_n^k} = \underline{q_i^k} - R_n^k \cdot \underline{q_i^n} \quad i \in (1,3) \quad (45)$$

På samme måte som i punkt 3.4, ble retningkosinmatrisa først funnet ved hjelp av hjelpevektorer og hjelpe-retningkosinmatriser. Da det er ønskelig å finne quadrokopterets posisjon og orientering i navigasjons ramma, måtte $\underline{q}_n^k$ endres til $\underline{q}_k^n$. Det ble gjørt på følgende måte:

$$\underline{q}_k^n = -R_k^n \cdot \underline{q}_n^k \quad (46)$$

Hvor:

$$R_k^n = (R_n^k)^T \quad (47)$$

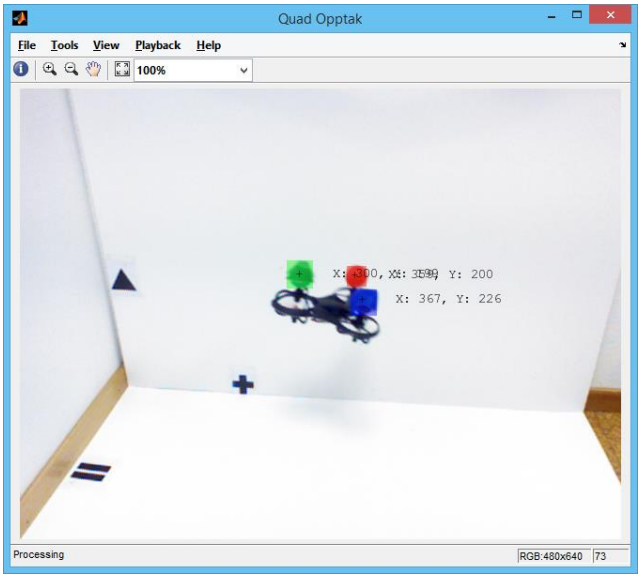
Fant da endelig basisen til quadrokopteret i navigasjonsramma på følgende måte:

$$\underline{p}_b^n = -(R_n^k)^T \cdot \underline{q}_n^k + (R_n^k)^T \cdot \underline{p}_b^k \quad (48)$$

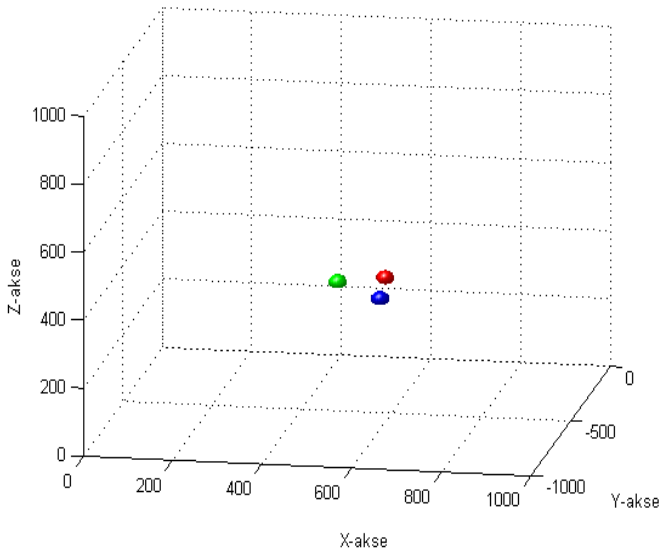
For å illustrere posisjonen til quadrokopteret i et 3d-plot, ble hver av markeringsballen transformert på følgende måte:

$$\underline{p}_i^n = -(R_n^k)^T \cdot \underline{q}_n^k + (R_n^k)^T \cdot \left(\underline{p}_b^k + R_b^k \cdot \underline{p}_i^b\right), \qquad i \in [1,3] \quad (49)$$

Figur 3-35, er et utklipp fra opptaket av quadrokopteret. Hvor symbolene som definerer n-ramma og markeringsballene på QK, er godt synlige. Figur 3-36, er quadrokoptert animert i navigasjons-ramma, basert på dataene fra Kinect.



Figur 3-35 Opptak av QK i n-ramma



Figur 3-36 Grafisk fremstilling av QK i n-ramma

Da quadokopteret har en intern regulering, vil prosjektets regulering kun basere seg på posisjon til QK i n-ramma og yaw-vinkelen. For å finne yaw-vinkelen det laget en hjelpvektor:

$$\underline{Y_b^n} = R_k^n \cdot R_b^k \cdot \begin{bmatrix} 100 \\ 0 \\ 0 \end{bmatrix} \qquad (50)$$

Deretter:

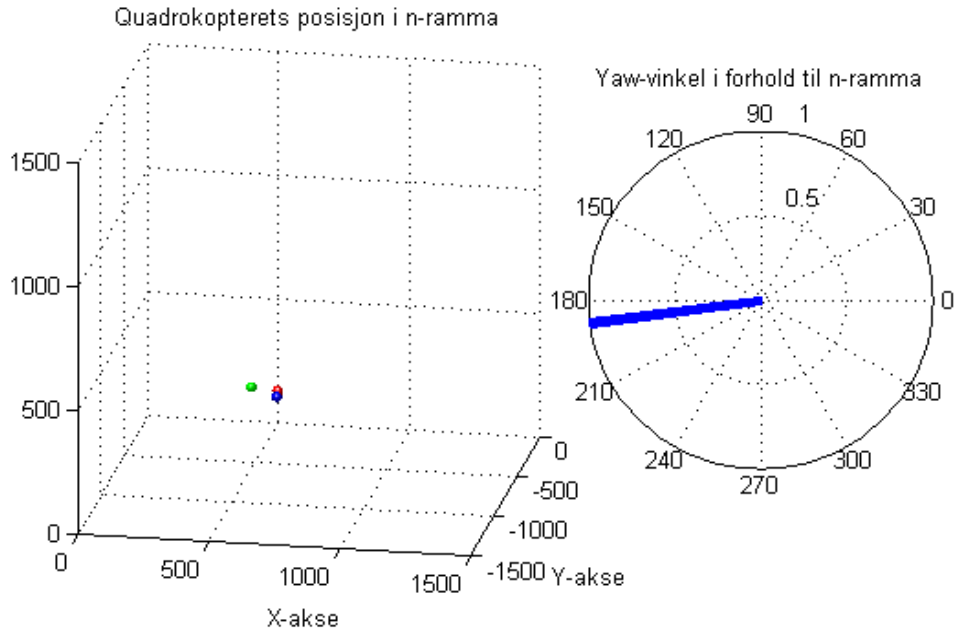
$$\phi_q^n = \cos^{-1} \frac{Y_b^n(x)}{\sqrt{Y_b^n(y)^2 + Y_b^n(x)^2}} \qquad (51)$$

Her er $\phi_q^n \in \pm[0, \pi]$. For å få ϕ_q^n til å være $\in \pm[0,2\pi]$, blir følgende logikk lagt til:

Pseudokode:

```
if Y_b^n(y)<0
  φ_q^n = π + (π − φ_q^n)
end
```

Fra Figur 3-37, er det tydelig hvor i navigasjonsramma og hvilke orientering quadrokopteret, har ved et tilfeldig valgt sample.



Figur 3-37 Posisjon og orientering i n-ramma

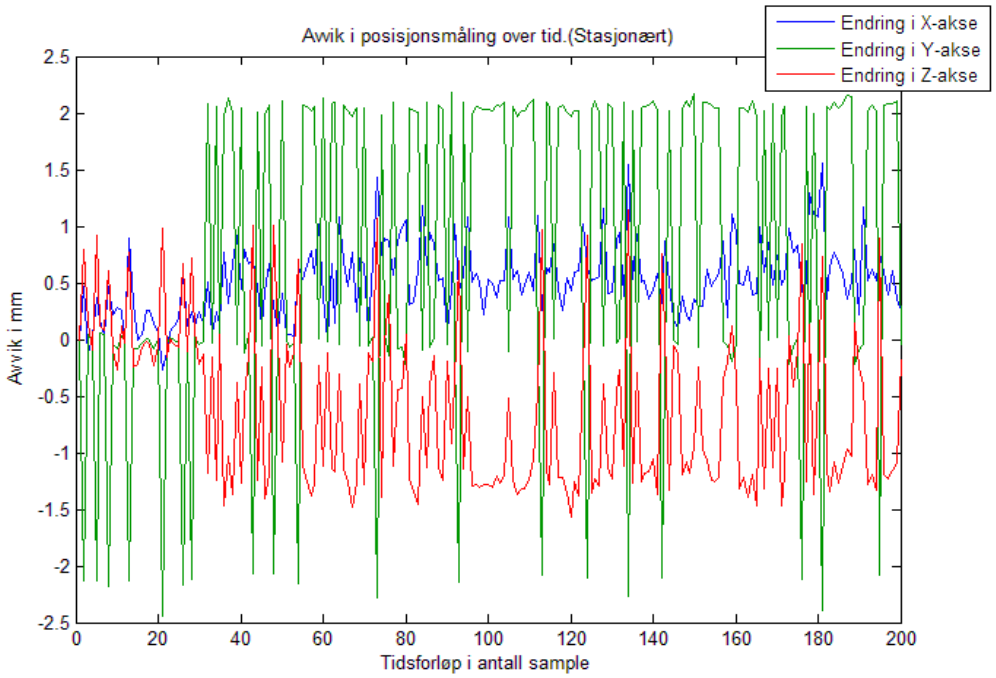
Symbolene som ble brukt for å markere navigasjons ramma, viste seg fungere dårligere enn antatt. Da Kinect sensoren var plassert ovenfor n-ramma, betyd det at symbolene ble projektert skjevt på bildeplanet i forhold til referansebildene. Det gjorde at den normaliserte krysskorrelasjonen ikke fant god match, som igjen betød at offseten ikke ble presis. Det som ble gjort for å kompensere for dette var å lage nye referansebilder. Dette ble gjort ved å ta bilde av symbolene som var påklistret i navigasjonsramma. På den måten ble korrelasjonen vesentlig forbedret fra ca. 0.6 til ca. 0.9.

I opptaket som ble benyttet for foregående data, viste det seg problematisk å kontrollere quadrokopteret innenfor navigasjonsrammen. En av grunnene var at QK var vesentlig lite i forhold til å løfte markeringsballene, se Figur 3-35, side 38. I tillegg til størrelsen var dette et quadrokopter av type Y4-oppsett, hvorav to av propellene i front spinner motsatt av hverandre. Dette har stor betydning for løft i det yaw-vinkelen blir justert. På grunn av dette ble kjøpt inn en ny quadrokopter av type x-copter. Denne var vesentlig større, 410 mm mot 170 mm, noe som gjorde at den ikke ble påvirket i samme grad av markeringsballene. I tillegg mister ikke denne høyde i det yaw-vinkelen blir justert.

3.4.6 Posisjonsendring ved støy

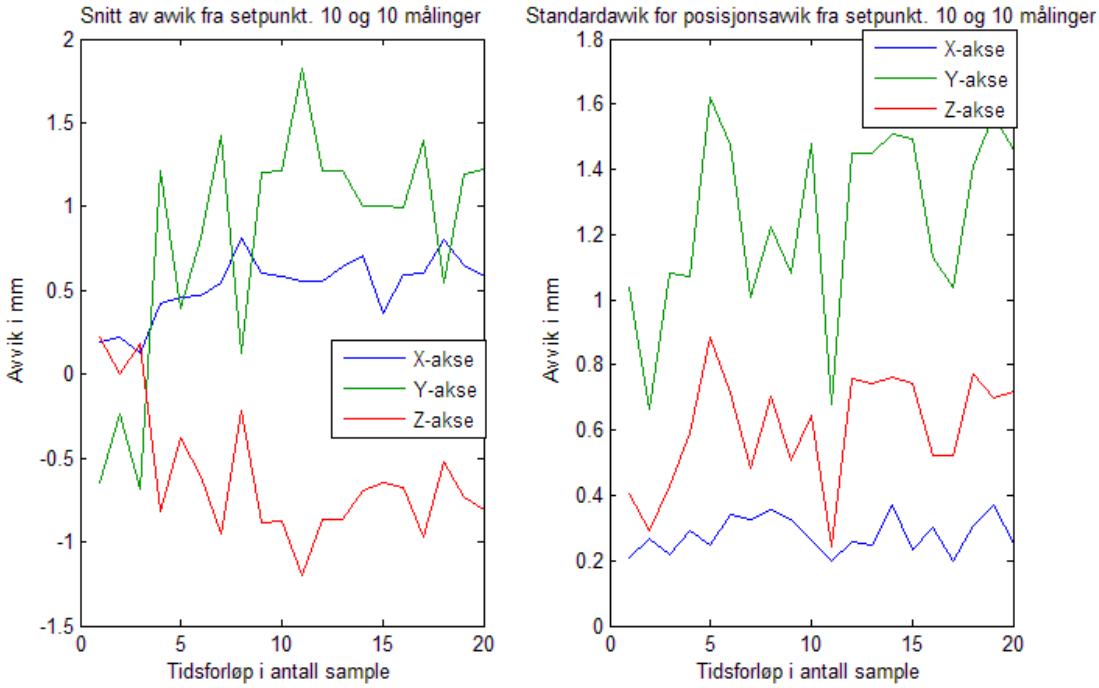
Da lysforholdet påvirker hvor godt Kinect observerer markeringsballene på quadrokopteret, vil den estimerte posisjonen til quadrokopter endre seg. Dette sees på som støy noe som ønskes å være så lite som mulig. For å finne denne støyen ble quadrokopteret plassert i navigasjonsrammen hvor det ble tatt en serie med målinger for å se hvor mye posisjonen endret seg over tid.

Ser av Figur 3-38, at det målte avviket fra settpunktet i navigasjonsramma ligger på ca. 0.5 mm i forhold til x-aksen, ± 1 mm for høyden(z-aksen) og ± 2mm for y-aksen.



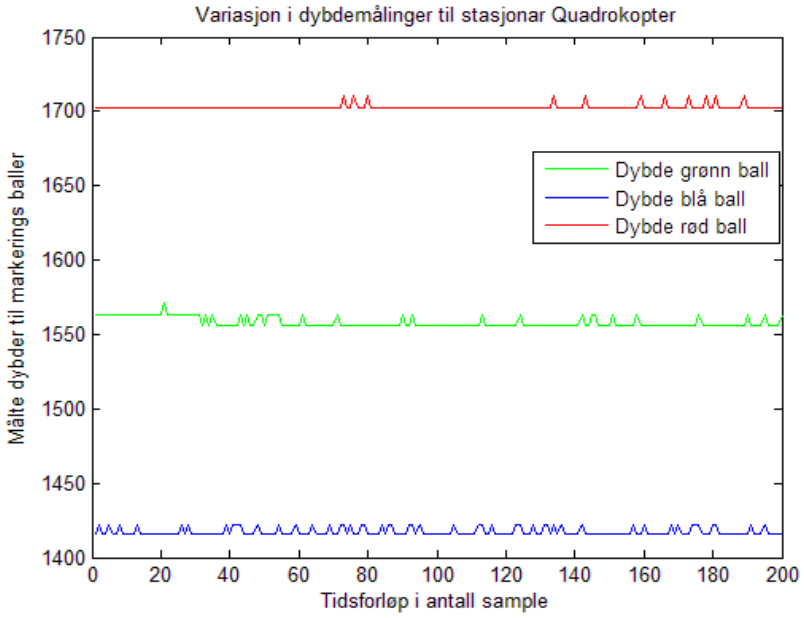
Figur 3-38 Avvik fra posisjon over tid

Figur 3-39 viser at standardavviket for posisjonen langs Y-aksen varierer vesentlig mer enn de andre aksene.



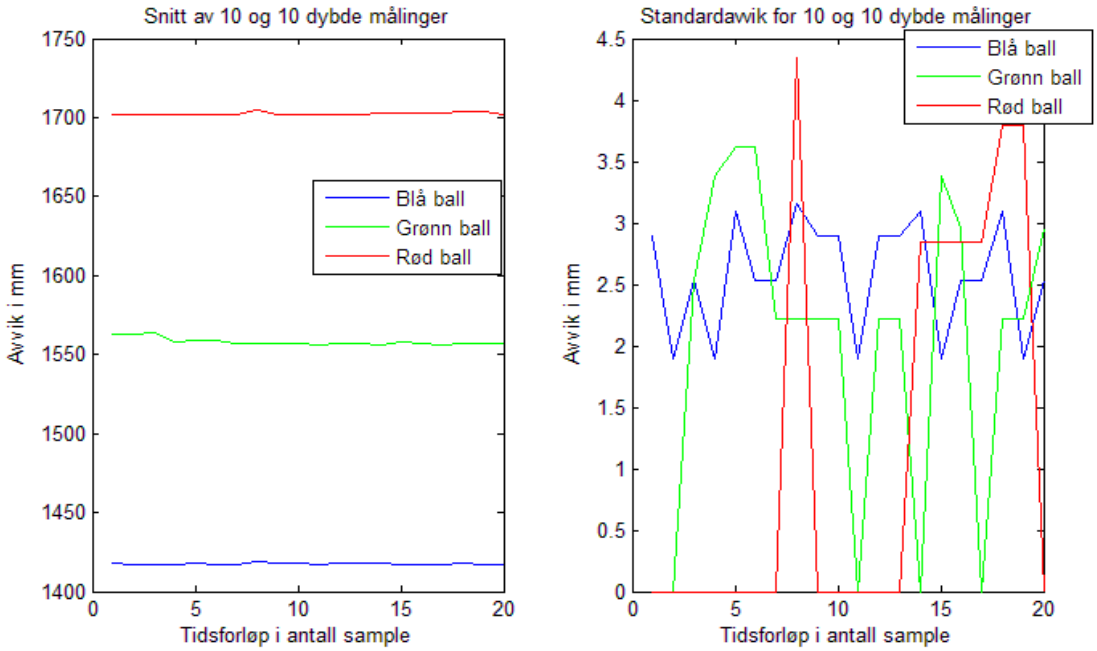
Figur 3-39 Midling og Std. hver 10. måling

Da posisjonen og orienteringen til quadrokopteret er vesentlig avhengig dybde-målingen, ble det også utført målinger på dybden til QK. Ser av Figur 3-40, at avstanden varierer vesentlig mer for blå og grønn markeringsball enn for den røde markeringsballen.



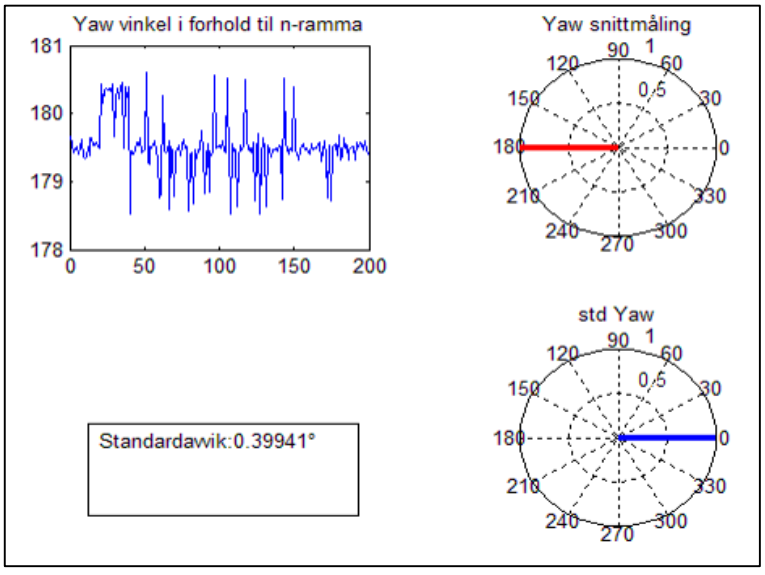
Figur 3-40 Dybdemålinger

I Figur 3-41, ser det ut til at standardavviket til ser ut til å variere mer enn de andre markeringsballene. Dette har med at den røde ballen var lengst unna Kinect, som betyr at når Kinect bommer på dybdemålingen, vil avviket være større enn ballen hadde vært nærmere.



Figur 3-41 Midling og std. av dybdemålinger

Stabiliteten av yaw-vinkelmålinger er vesentlig for er autopilot. Målinger over tid vise følgende data:



Figur 3-42 Yaw-vinkel målinger

Ut i målingene er det tydelig at objekt-deteksjonen for fargebilde har vesentlig påvirkning på dybdemålingene og da spesielt deteksjonen av den blå- og grønne markeringsballene. Da posisjonen til hver av markerings ballene blir estimert, er dette origo til ballene. En ball er åpenbart rund, noe som betyr at avstanden til ballen vil variere litt avhengig av hvor på ballen avstanden blir målt til. Posisjon- og yaw-målingene ble viste seg å variere lite og være tilfredsstillende for en autopilot i denne omgang.

3.4.7 Nøyaktigheten i posisjon

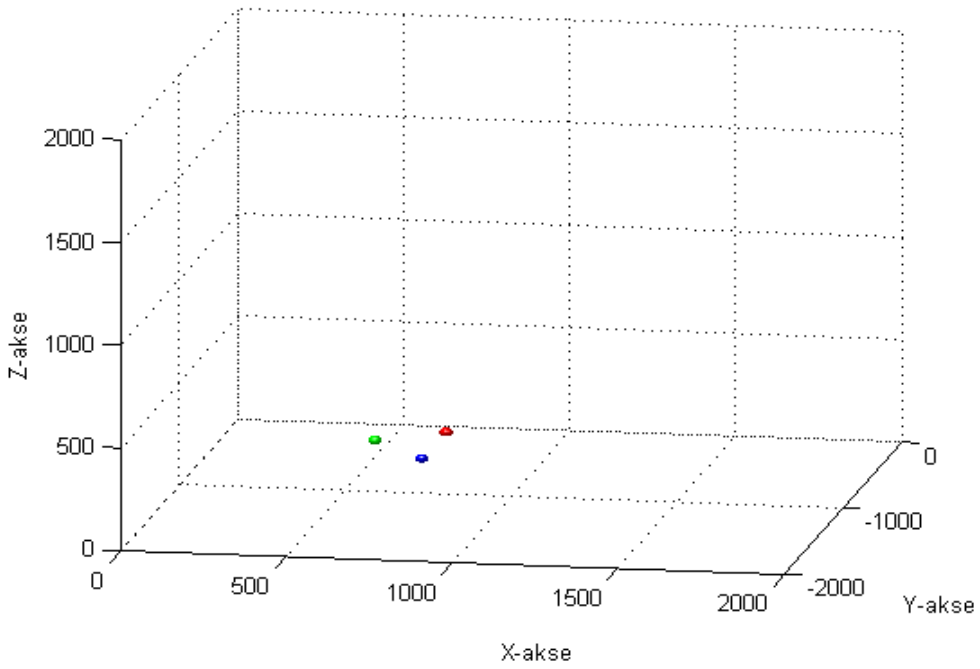
Quadrokopteret ble plassert i punktet, TP i navigasjons ramma. Det ble gjort målinger over enn periode hvor gjennomsnittet til posisjonen ble som følger:

Ønsket posisjon (mm): $$TP = \begin{bmatrix} 600 \\ -300 \\ 0 \end{bmatrix}$$

Målt posisjon (mm): $$Quad\_center = \begin{bmatrix} 585{,}8 \\ -319{,}1 \\ -21{,}3 \end{bmatrix}$$

Differanse (mm): $$diff = \begin{bmatrix} 14{,}2 \\ 19{,}1 \\ 21{,}3 \end{bmatrix}$$

Feilen(diff) var på rundt 2 cm, noe som vil være akseptabelt da nøyaktigheten til plasseringen av QK i TP ble gjort for hånd og kunne være noe upresis. Da målene til navigasjonsramma er basert på krysskorrelasjon, vil dette også bidra til usikkerhet. Figur 3-43 viser hvor quadrokoteret var estimert til å være i navigasjons ramma.



Figur 3-43 QK i navigasjons ramma

4 Autopilotdesign og uttesting

En autopilot er et system som blir brukt for å kontrollere et fartøy i en gitt bane uten menneskelig interaksjon. Autopiloten må på forhånd ha lastet inn en oppgave/bane foruten reguleringen. Det er vanlig å lage simuleringer av både fartøyet og autopiloten før autopiloten implementeres, men grunnet begrenset tid ble det valgt å lage autopilot og teste denne direkte på quadrokopteret uten å teste denne på noen simulering først. For å regulere QK var det nødvendig med et grensesnitt mellom PC og fjernkontrollen.

4.1 Oppkobling til fjernkontroll

For å benytte den kommersielle innkjøpte quadrokopteret var det nødvendig å gjøre noen modifiseringer på fjernkontrollen. Kontrollsignalene til hvert av kanalene på fjernkontrollen ble originalt justert av potmetre (regulerbar motstand), det vil si at det var den elektriske spenningen som varierte pådragene. For å finne disse spenningene ble disse målt.

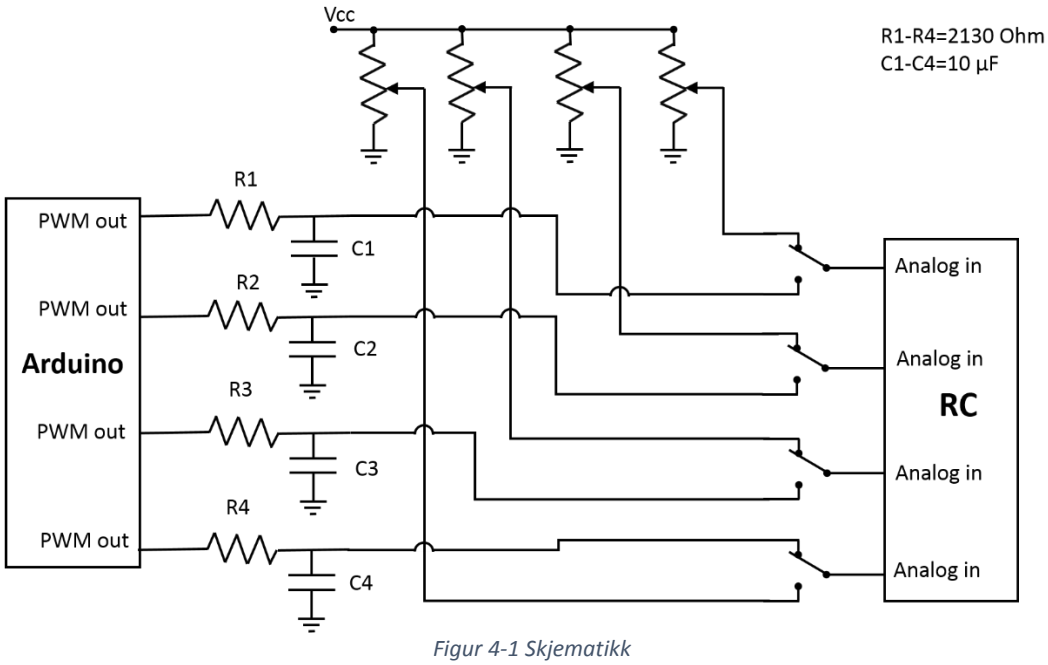
Måledata viste:

Tabell 4-1 Fjernkontroll målinger

	Min (volt)	Senter (volt)	Maks (volt)
Pådrag	0,32	1,72	3,12
Yaw	0,2	1,72	3
Pitch	0,96	1,84	2,72
Rull	0,84	1,6	2,52

Spenningene viste seg å variere fra kanal til kanal. For å erstatte potensiometrene og hvert av kanalene ble det valgt å bruke en mikrokontroller da denne har muligheten til å modulere analoge signaler ut(PWM).

Figur 4-1, viser koblingen som ble gjort mellom mikrokontrolleren og fjernkontrollen.



Figur 4-1 Skjematikk

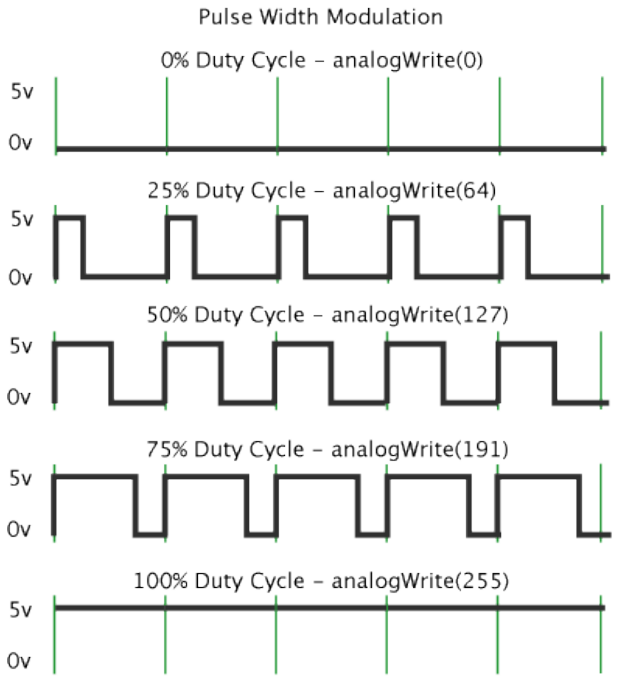
I skjematikken er det RC-ledd etter hver av digitalutgangene til mikrokontrolleren. Disse ble lagt til for å glatte ut PWM signalene for å likespenning, DC. Det ble også koblet til brytere til hver av kanal, slik at det var en mulighet å fortsatt benytte de originale kontrollstikkene.

Valget av mikrokontroller ble en Arduino Mini Pro, da dette var en mikrokontroller som prosjektet allerede var i besittelse av. Arduino Pro Mini er et mikrokontroller-brett som baserer seg på ATmega168 eller ATmega328P, hvor det var sistnevnte som ble benyttet.

For å skrive ut PWM signaler, gjøres dette ved å sette bit-verdier. Arduino operer med 8-bits verdier for å skrive ut PWM signalene. Det vil si at bit-verdi 255, vil være 5 volt og bit-verdi 0 er jord, altså 0 volt. Siden fjernkontrollen fikk spenningensverdier fra ca.0-3,3 volt, måtte det skrives verdier fra 0-168.

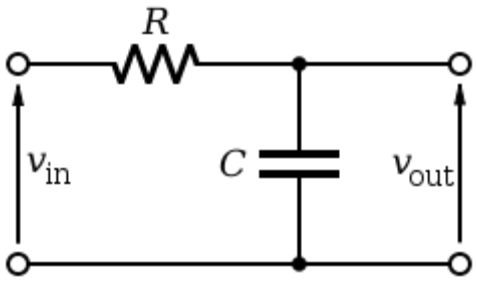
4.1.1 PWM signal til analog ut

Arduino Mini Pro har 4 analoge innganger, men har ingen analoge utganger. Den kan derimot modulere PWM signaler og sende disse ut på digitalutgangene. Mikrokontrolleren modulerer PWM signalene med to frekvenser, 490- og 980 Hz. Dette har riktignok ingen betydning for dette formålet da det kun er forholdet mellom høyt og lavt signal som avgjør spenningen ut. Figur 4-2, illustrerer prinsippet med PWM signaler.



Figur 4-2 PWM Signal

For å glatte ut PWM signalene til DC-spenninger, ble det laget et RC-lav pass filter til hver av utgangene. Se Figur 4-3 RC-filter.



Figur 4-3 RC-filter

Da det kun var ønskelig med DC-spenning, måtte filteret ha en lav «cut off» frekvens. På den måten ville PWM signalet bli filtrert ut og gi en jevn DC spenning ut. For å finne kondensator- og motstands-verdi som kunne benyttes, utledes følgende:

Kirchhofs 2.lov sier at:

$$V_{in}(t) = V_R(t) + V_{out}(t) \quad (52)$$

Og siden fjernkontrollen kun skal «lese» spenneningen, tas det utgangspunkt i at det ikke går strøm igjennom fjernkontrollen. Ved hjelp av Ohms lov, fås:

$$V_{in}(t) = V_{out}(t) + R \cdot I(t) \quad (53)$$

Ladningen i kondensatoren som blir lagret over tid:

$$Q_C(t) = C \cdot V_{out}(t) \quad (54)$$

Strømmen over tid er gitt av endringene i ladningen til kondensatoren. Gitt på følgende måte:

$$I(t) = \frac{dQ_C}{dt} \quad (55)$$

Setter likning (55) inn likning (54), får da:

$$I(t) = C \cdot \frac{dV_{out}(t)}{dt} \quad (56)$$

Setter linking (56) inn i likning (53):

$$V_{in}(t) = V_{out}(t) + R \cdot C \cdot \frac{dV_{out}(t)}{dt} \quad (57)$$

For å løse denne 1. ordens differensiallikningen, brukes Laplace transformasjon:

$$V_{in}(s) = V_{out}(s) + R \cdot C \cdot sV_{out}(s) \quad (58)$$

Løser ut for $V_{out}(s)$ og får:

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{1}{1 + s \cdot R \cdot C} \quad (59)$$

$$H(s) = \frac{1}{R \cdot C \cdot s + 1} \quad (60)$$

Da dette er et 1. orden transferfunksjon, vil variabelen tilhørende s, være tidskonstanten τ. Får da:

$$H(s) = \frac{1}{\tau \cdot s + 1} \quad (61)$$

Siden $\tau = \frac{1}{\omega_b}$, og hvor $\omega_b = 2\pi f_b$ blir følgende:

$$R \cdot C = \frac{1}{2\pi f_b} \quad (62)$$

Sammenhengen mellom tids forsinkelse, τ, og knekkfrekvens, ω_c, i forhold til RC leddet er gitt:

$$\tau = RC \quad (63)$$

Ved store kondensator(C)- og motstands(R)-verdier, kan «cut-off» frekvensen settes så lav som mulig. Men det er en «trade-off». Ved å velge store motstandere og store kondensatorer, vil tidskonstanten bli veldig høy. Det vil si at impulsresponsen blir treg. Ved lave RC verdier, øker «cut-off» frekvensen hvor det i tillegg vil bli mye «ripple» på utgangssignalet.

Fra testene hvor posisjonen og orienteringen ble estimert kapittel 3.4, ble det målt at tiden det tok å kjøre igjennom programmet lå rundt 0.2 sekunder. Det var derfor ikke kritisk å velge lave RC verdier for å få ned responstiden til filteret. RC-verdiene ble da som følger:

$$R = 2130\Omega$$
$$C = 10\mu F$$

Løser likning (62) av hensyn på f_b og får:

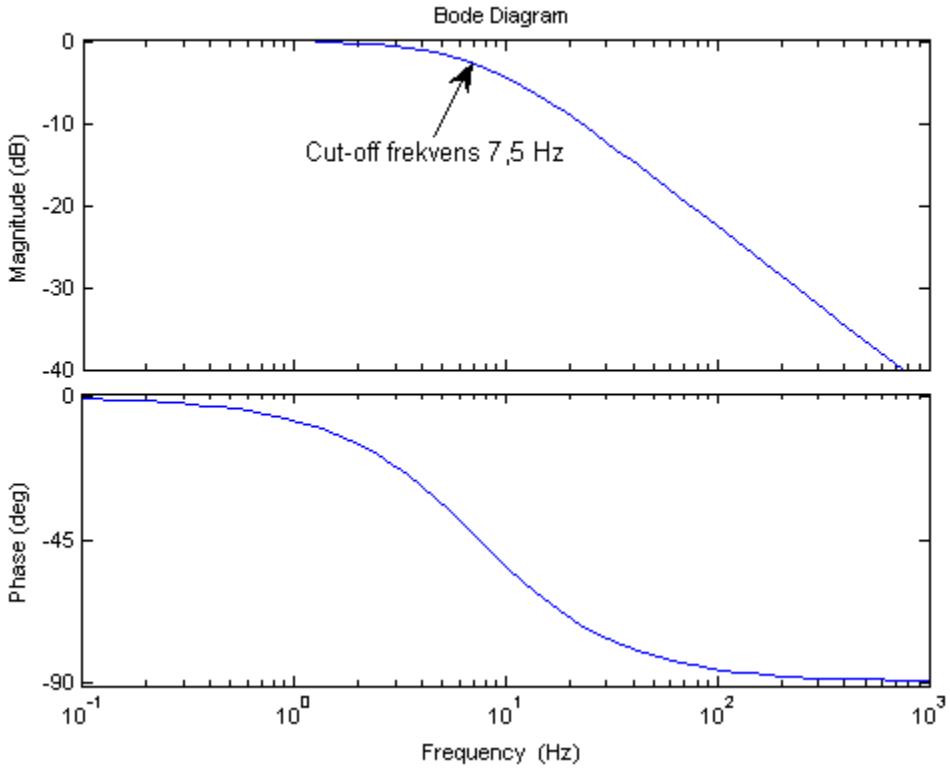
$$f_b = \frac{1}{2 \cdot \pi \cdot 2130\Omega \cdot 10\mu F} \approx 7{,}5\ Hz$$

Fra likning (63) fås:

$$T = 2130\Omega \cdot 10\mu F = 0{,}0213s$$

Tidskonstanten er da ca. 1/10 del av tiden det tok for PC-en å kjøre igjennom programløkka.

Ser av Figur 4-4 Bodeplott, at knekk-frekvensen ligger på 7,5 Hz. Dette er da langt under den laveste frekvensen til PWM moduleringen som var 490 Hz.



Figur 4-4 Bodeplott

Hvis programkoden skal kjøres på raskere PC-er, vil det være en mulighet å justere ned RC-konstantene. Til denne oppgaven var disse parameterne tilstrekkelig.

4.1.2 Kommunikasjon til mikrokontroller

For å sende pådragsdata fra autopiloten/Matlab til mikrokontrolleren, ble det valgt å sende seriell data da begge deler støttet dette.

Seriell data, RS232, gjennom en COM-port er en enkel metode å overføre data på. Som navnet antyder blir data sendt serielt et bit av gangen. Seriell data blir fortsatt mye brukt til å programmere og kommuniserer med industriutstyr som rutere, mikrokontrollere, etc., men dette er en aldrende måte å sende data. Det er det derfor en sjelden port å finne på nyere datamaskiner og da spesielt på bærbare. Det var derfor nødvendig med en USB-til-seriell adapter.

Da autopiloten kun skulle sende kommandoer til fjernkontrollens gass-pådrag, yaw, pitch og rull, er det lite data som skulle overføres. Det holdt derfor med en standard seriell overføringshastighet på 9600 baud/sekund. Siden seriell dataoverføring er en veldig enkel dataoverføring, vil ikke PC og mikrokontroller være synkroniserte. Det vil si at pådragskommandoer til, eksempelvis, rull kunne registreres som pådrag til pitch. Dette ble løst med å sende starts-verdi og slutt-verdi, før og etter pådrags kommandoene.

Da maks spenningen til fjernkontrollen var på ca. 3.3 volt, 168 i bit-verdi, var dette verdier som regulatoren ikke ville sende til hvert av kanalene til fjernkontrollen. Startverdien ble derfor valgt til å være: 250, og sluttverdien ble valgt til å være: 240.

Mikrokontrolleren venter da til den har fått startverdien, før den registrerer dataene til hver kanal, før den mottar stopp-verdien. Deretter ble PWM signalene modulert basert på disse dataene.

Eksempel på en slik datastrøm han være:

Tabell 4-2 Eksempel datastrøm

250	15	65	91	120	240

For å kontrollere at mikrokontrolleren registrerte de riktige verdiene, ble det gjort en test hvor mikrokontrolleren inkrementerte mottatte verdier med 1, for deretter å sende dette serielt tilbake til Matlab, men da uten start- og stopp-verdier.

Resultatet ble som følger:

Tabell 4-3 Sendt testdata

250	10	20	30	40	240

Mottatt data:

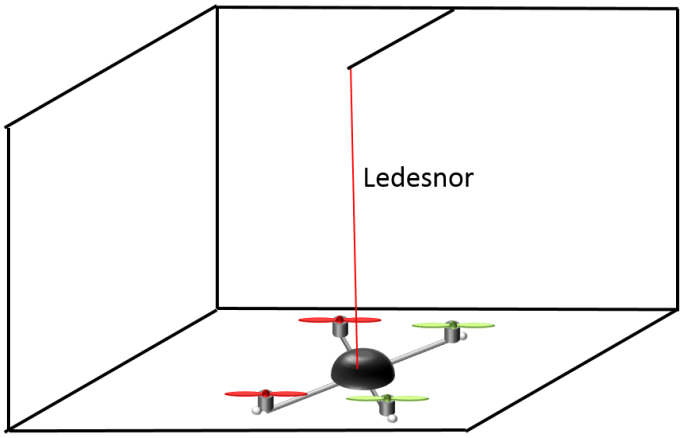
Tabell 4-4 Mottatt testdata

21	31	41	11

Resultatet ble at Matlab fikk en annen tallrekkefølge enn det den hadde sendt. Dette betydde da at også sending av data tilbake til Matlab, var usynkronisert. Dette kunne blitt løst ved å la Matlab vente på start- og stopp-verdier, men dette ville da ha gått utover prosesseringstiden.

4.2 Autopilot

For å lage scenarioet til autopiloten, ble settpunktet satt til 200 mm over utgangsposisjonen til quadrokopteret. Til dette oppsettet ble et quadrokopter plassert ca. i midten av navigasjonsramma med en ledesnor tredd igjennom quadrokopter-skroget. På denne måten var det mulig å teste og justere regulatoren til pådraget uten at QK driftet ut av n-ramma. Oppsettet ble som Figur 4-5.



Figur 4-5 QK oppsett i n-ramma

4.2.1 Regulering av gass-pådrag

For å lage reguleringen til gass-pådraget ble det i første omgang laget en P-regulator. P-regulatorer ser kun på avviket mellom målt posisjon og settpunkt. Dette avviket blir deretter multiplisert med en faktor K_p, en forsterkning. Dette er en forsterkning som er unik fra system til system. For prosjektets del var denne forsterkningen basert på at spenningsverdiene som fjernkontrollen benyttet. Det betød at det endelige gass-pådraget ikke måtte overskride verdien 168. Algoritmen ble da som følger:

Pseudo kode:
pos_error = sett_punkt-målt_pos
gass_pådrag = K_p · pos_error

if (gass_pådrag > 168)
gass_pådrag=168

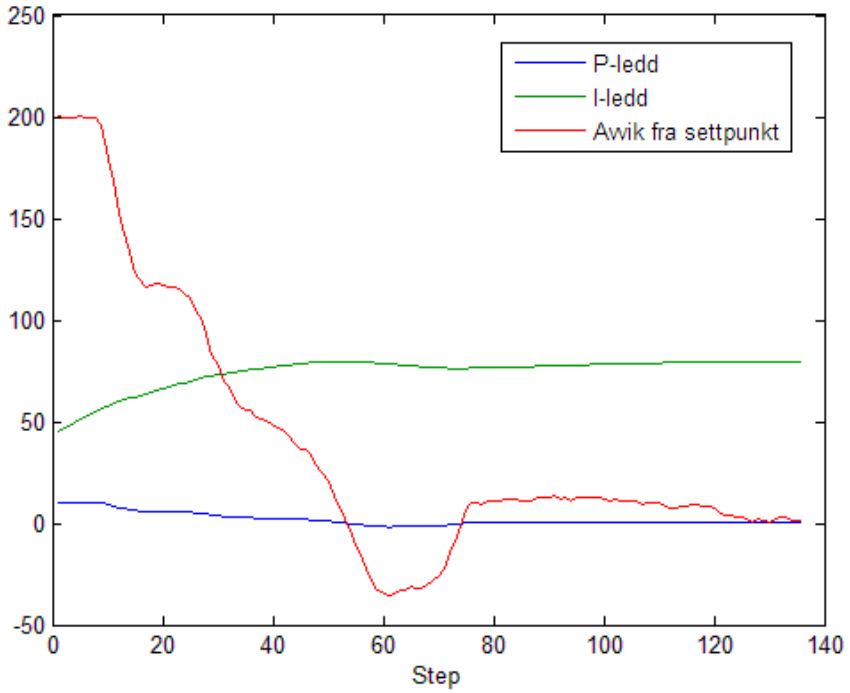
Forsterkningen K_p, ble gradvis økt til quadrokopteret begynte å oscillere, for deretter å bli nedjustert. Det vil si at QK aldri ville nå settpunktet, kun med et P-ledd. For å løse dette ble det lagt til et integrator-ledd. Et integrator-leddet summerer opp feilen over tid. Har i tillegg en forsterkning, K_i. Algoritmen bla da som følger:

Pseudo kode:
I_ledd=I_ledd + K_i · pos_error
Gass_pådrag=P_ledd+I_ledd

if (gass_pådrag > 168)
gass_pådrag=168

Forsterkningen til integrator-leddet ble satt lavt, for deretter økt til quadrokopteret oversvinget mer enn hva som var rimelig.

Figur 4-6, viser hvordan avviket minker i takt med at integrator-leddet øker og proposjonal-leddet minker



Figur 4-6 Regulering av gass-pådrag

Det er tydelig ifra figuren at avviket fra settpunktet går i trappetrinn. Dette var grunnet ledesnoren som var tredd igjennom skroget til quadrokopteret, hektet seg opp ved jevne mellomrom.

Da prosesseringen viste seg å vær tung ble PC 'en vesentlig varm. For å hindre eventuelle skader på CPU, justerer PC'en ned prosessorkraften ved jevne mellomrom. Resulterte ble at prosesseringen av bildedata og regulerings løkka brukte ulik tid for hver gjennomkjøring. På denne måten ble pådraget, basert en måling, uforandret over lengre tid. Dette resulterte i at quadrokopteret oscilerte om settpunktet. For å kompensere for dette ble et variabelt pådragsledd lagt til proposjonal-leddet i PI-regulatoren. Dette leddet var et forholdstall basert på gjennomsnittet av tiden det tok å kjøre gjennom prosesserings løkka da PC'en var kald. Gjennomsnitts tiden ble målt til: 0,2060 sekunder.

Ut i fra dataene ble snittet av målingene brukt som referanse. Proporsjonal-leddet ble deretter som følger:

$$P = K_p \cdot pos_{error} \cdot \frac{t}{T_{ref}}$$

Med dette forholds-leddet ble P-leddet mindre «aggressiv» og integrator leddet fikk mer av jobben av å nå settpunktet. Reguleringen av gass-pådraget viste seg å være rask og presis, og klarte å holde settpunktet nøyaktig.

5 Konklusjon

I denne oppgaven er det undersøkt hvor godt Microsofts Kinect kan erstatte GPS for innendørs treghetsnavigasjon for Quadrokoptere. Det har blitt gjort kalibreringer av Kinect med en Matlab Toolbox utviklet av Jean-Yves Bouguet, og en mer manuell kalibrering. Det har blitt sett på hvilke funksjoner som kan benyttes for å søke etter objekter av interesse i et bilde. Og det har blitt gjort posisjons- og orienterings-estimasjon av quadrokopter med tilhørende støydata. Deretter har det blitt laget et grensesnitt mellom autopilot og fjernkontroll til QK. Til slutt ble det laget og testet en autopilot med en enkel bane til et quadrokopter.

For scener dypere enn 1,5 meter, viste det seg at den manuelle kalibreringen fungerte godt for å finne dybden til et objekt i et fargebilde direkte, med de samme pikselkoordinatene. For scener som er nærmere, anbefales det å benytte koordinattransformasjon basert på kalibreringsdataene.

Under gode lysforhold viste det seg å fungere godt med prosjektets algoritmer for å finne og segmentere markeringsballene til quadrokopteret. Posisjonsestimeringen viste seg å være tilfredsstillende for en autopilot og grensesnittet mellom PC og fjernkontrollen fungerte som det skulle. Det største problemet var prosesseringshastigheten til PC/Matlab. Da denne var veldig treg, 0.2 sekunder, og enda tregere når den ble varm, viste det seg vanskelig å lage sanntids reguleringen til pitch, rull og yaw. I tillegg til prosesseringshastighet, var det vesentlig liten oppløsning på fjernkontrollen som gjorde småjusteringer til pådragene, vanskelig.

5.1 Videre arbeid

Matlab er ikke spesielt egnet for sanntid prosessering. Til eventuelle senere prosjekter vil det anbefales å se på alternative metoder å prosessere dataene. Matlab har muligheten til å opprette såkalte GPU-array for bildeprosessering. Dette er en metode for å prosessere dataene parallelt, men krever et NVIDIA skjermkort(GPU). Det er heller ikke alle funksjoner som støtter GPU-array. Det anbefales også å finne ut hvilke muligheter det er i simulink, da denne skal være bedre til sanntid prosessering.

For å gjøre deteksjonen til markeringsballene noe mer robust ovenfor lysforhold, kan dette i første omgang gjøres med et Kalman-filter basert på bilde. Matlab Vision System Toolbox, har innebygde funksjoner for denne type estimering. En annen måte å gjøre gjenkjenningen av markeringsballene mer robust på, vil være å ha markerings punkter som er vesentlig tydeligere i bildet, da gjerne lysdioder.

Det anbefales også å benytte et quadrokopter med høyere oppløsning på kontrollstikkene, og med mulighet for telemetri. På den måten kan sensor data sende trådløst til PC, som igjen kan benyttes til posisjons og orienterings estimering. Hvis quadrokopteret i tillegg er utstyrt med et magnetometer, et digitalt kompass, vil det ikke være behov for markeringspunkter med forskjellige farger. Så lenge markeringspunktene er plassert slik at orienteringen til quadrokopteret kan registreres før noen flyvning finner sted.

Neste steg vil være å videreutvikle autopiloten og lage en banegenerator til denne.

6 Referanser

[1] J. G. Leishman, «A History of Helicopter Flight,» [Internett]. Available: http://terpconnect.umd.edu/~leishman/Aero/history.html.

[2] J. M. a. J. Yeazel, «gpsinformation.net,» [Internett]. Available: http://gpsinformation.net/main/gpspower.htm.

[3] Several, «Open Kinect,» [Internett]. Available: http://openkinect.org/wiki/Main_Page.

[4] E. W. Weisstein, «--A Wolfram Web Resource,» MathWorld, [Internett]. Available: http://mathworld.wolfram.com/VectorSpace.html.

[5] E. W. Weisstein, «A Wolfram Web Resource,» MathWorld, [Internett]. Available: http://mathworld.wolfram.com/AffineSpace.html.

[6] E. Ackerman, «http://spectrum.ieee.org/,» 9 5 2011. [Internett]. Available: http://spectrum.ieee.org/automaton/robotics/industrial-robots/quadrotor-formation-flying-gets-aggressive.

[7] K. K. *. a. S. O. Elberink, «Accuracy and Resolution of Kinect Depth Data for Indoor,» *MDPI - Open Access Publishing ,* p. 18, 2012.

[8] «Developer Network-Kinect for Windows Sensor,» Microsoft, [Internett]. Available: http://msdn.microsoft.com/en-us/library/hh855355.aspx. [Funnet Januar 2014].

[9] «www.mathworks.se,» MathWorks, [Internett]. Available: http://www.mathworks.se/hardware-support/kinect-windows.html.

[10] J.-Y. Bouguet, «Camera Calibration Toolbox for Matlab,» [Internett]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/index.html#parameters.

[11] J.-Y. Bouguet, «Corner extraction, calibration, additional tools,» [Internett]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html.

[12] «arduino.cc,» [Internett]. Available: http://arduino.cc/en/Guide/ArduinoProMini. [Funnet Januar 2014].

[13] «arduino.cc,» [Internett]. Available: http://arduino.cc/en/Main/ArduinoBoardProMini. [Funnet Januar 2014].

[14] «msdn.microsoft.com,» Microsoft, [Internett]. Available: http://msdn.microsoft.com/en-us/library/hh973075.aspx. [Funnet Januar 2014].

[15] I. A. Toolbox, «Mathworks,» [Internett]. Available: http://www.mathworks.se/products/imaq/.

[16] C. V. S. Toolbox, «Mathworks,» [Internett]. Available: http://www.mathworks.se/products/computer-vision/.

Appendiks A: Matlab koder

Appendiks A-1: Matlab kode-Bilde differanse

```
%############## Endringsdeteksjon av 2 bilder ##############################

%   En test for å detektere differansen mellom 2 bilder. For deretter å
%   filtere ut og filtere ut støy
%#########################################################################
clc
clear all;
close all;

obj = VideoReader('quad_pic_0001.avi'); %Henter opptak
vid = read(obj);
frames = obj.NumberOfFrames;

%Da dette opptaket var en .avi-fil, må denne gjøres om til 2 hånterbare
%bilder.

for x = 1 : frames
    imwrite(vid(:,:,:,x),strcat('frame-',num2str(x),'.tif'));
    orgpic=imread(['frame-' num2str(x) '.tif']);
    quad{x}=imresize(rgb2gray(orgpic),0.25);

end
%%
diff=imabsdiff(quad{1},quad{2});
figure
imshow(diff);

tresh=graythresh(diff);     %Fremhever det grå
bw=(diff >= tresh*255);
figure;
imshow(bw);

%Fjerner pixsler som er mindre enn 10 pixler
bw2 =xor(bwareaopen(bw, 10), bwareaopen(bw, 50));
figure;
imshow(bw2);
hold on

%Finner "alt" av mål av objektene på bilde
props=regionprops(bw2,'all');

h=rectangle('Position',props(1).BoundingBox);
test=props(1).BoundingBox;
set(h,'edgecolor',[1 0 0]);

plot(props(1).Centroid(1),props(1).Centroid(2),'go','linewidth',2,'markersize',10);

x=props(1).ConvexHull(:,1);
y=props(1).ConvexHull(:,2);
plot(x,y,'g');
%%
border=([props(1).Centroid(1)*4, props(1).Centroid(2)*4]);
figure;
imshow(orgpic);
hold on
viscircles(border, 90,'EdgeColor','b');
figure;
newpic=imcrop(orgpic,[border(1) border(2) props(1).Area(1)*4 props(1).Area(1)*4]);
imshow(newpic);
```

Appendiks A-2: Deteksjon og dimensjonsmåling av QK

```
%############### Deteksjon og dimensjonsmåling av QK  ##################

% Dette programmet laster inn et testbilde av et QK,detekterer
% markeringsballer og finner posisjon med måledata

%#########################################################################
clc
clear all;
close all;

load('rgbFrame.mat')
load('depthFrame.mat')

redThresh=0.20;
greenThresh=0.05;
RadPingBall=20;% Radius til bordtennisball

p1b=[50;-50;0];
p2b=[-50;-50;0];
p3b=[50;50;0];
p4b=[-50;50;0];
             %Søker i bilde etter objekter
obj_det = vision.BlobAnalysis('AreaOutputPort', false, ...
                              'CentroidOutputPort', true, ...
                              'BoundingBoxOutputPort', true', ...
                              'MinimumBlobArea', 100, ...
                              'MaximumBlobArea', 3000, ...
                              'MaximumCount', 4);
redCircleFlag=0; greenCircleFlag=0; rgbPic=rgbFrame; depthPic=depthFrame;

diffFrameRed=rgbPic(:,:,1)-((rgbPic(:,:,1)+rgbPic(:,:,2)+rgbPic...
    (:,:,3))/3);
binFrameRed = im2bw(diffFrameRed, redThresh);

diffFrameGreen=rgbPic(:,:,2)-((rgbPic(:,:,2)+rgbPic(:,:,1)+rgbPic...
    (:,:,3))/3);
binFrameGreen = im2bw(diffFrameGreen, greenThresh);

[greenBallsCenter,greenArea]=step(obj_det,binFrameGreen);
[redBallsCenter, redArea]=step(obj_det,binFrameRed);

%en ekstra sjekk om det to markeringskuler i bildet.
if size(greenBallsCenter,1)<2
    [greenBallsCenter,greenBallsRadi]=imfindcircles(binFrameGreen,[10 20]);
else
    greenBallsRadi=((greenArea(:,3)+greenArea(:,4))/4);

end

if size(redBallsCenter,1)<2
    [redBallsCenter,redBallsRadi]=imfindcircles(binFrameRed,[10 20]);
else
    redBallsRadi=((redArea(:,3)+redArea(:,4))/4);

    redCircleFlag=1;
end

diffImage=imabsdiff(binFrameGreen,binFrameRed);
[blobdiff,bounddiff]=step(obj_det,diffImage);

if ~isempty(bounddiff)
diffImageCut= diffImage(min(bounddiff(:,2)):max(bounddiff(:,2))...
    +max(max(bounddiff(:,3:4))),min(bounddiff(:,1)):max(bounddiff(:,1))...
```

```
    +max(max(bounddiff(:,3:4))));
end

%Sjekker korrelasjonen mellom rgb bilde og dybde bilde og finner evt.offset
depthPic(depthPic>max(max(depthPic))-150)=0;
cc = normxcorr2(diffImageCut,logical(depthPic));
   [max_cc, imax] = max(abs(cc(:)));
   [ypeak, xpeak] = ind2sub(size(cc),imax(1));
   corr_offset = [ (ypeak-size(diffImageCut,1))...
       (xpeak-size(diffImageCut,2)) ];

   if max_cc >= 0.5         %Korrelasjonen >= 0.5,er "match"
       greenDepthCord=zeros(size(greenBallsCenter));
       for i=1:size(greenBallsCenter,1)
        greenDepthCord(i,:)=[(greenBallsCenter(i,2)...
            -min(bounddiff(:,2)))+corr_offset(1),...
            (greenBallsCenter(i,1)-min(bounddiff(:,1)))+corr_offset(2)];
       end

       redDepthCord=zeros(size(redBallsCenter));
       for i=1:size(redBallsCenter,1)
       redDepthCord(i,:)=[(redBallsCenter(i,2)...
           -min(bounddiff(:,2)))+corr_offset(1),...
           (redBallsCenter(i,1)-min(bounddiff(:,1)))+corr_offset(2)];
       end
   end

if ~isempty(greenBallsRadi)
    mmradiusGreenBalls=zeros(2,1);
    depthGreenBall=zeros(2,1);

        for i=1:size(greenBallsRadi,1)
            depthGreenBall(i)=double(depthPic(greenDepthCord(i,1),...
                greenDepthCord(i,2)));

            mmradiusGreenBalls(i)=greenBallsRadi(i)*...
                (depthGreenBall(i)*0.001933);

            if  (mmradiusGreenBalls(i) > (RadPingBall-5))...
                    && (mmradiusGreenBalls(i) <= (RadPingBall + 5))...
                    && (size(greenBallsRadi,1)>1)%Litt slingrings mon
                %som referanse til å finne orienteringen i X,Y planer
                if i==2
                    dist_green_Y=abs(greenBallsCenter(i-1,2)-...
                        greenBallsCenter(i,2))*(min(depthGreenBall)...
                        *0.001933); %Bruker den nærmeste kula
                    dist_green_X=abs(greenBallsCenter(i-1,1)-...
                        greenBallsCenter(i,1))*(min(depthGreenBall)...
                        *0.001933);
                end
            end
        end
  end

if ~isempty(redBallsRadi)
    mmradiusRedBalls=zeros(2,1);
    depthRedBall=zeros(2,1);

    if redCircleFlag==0
        for i=1:size(redBallsRadi,1)
            depthRedBall(i)=double(depthPic(redDepthCord(i,1),...
                redDepthCord(i,2)));

            mmradiusRedBalls(i)=redBallsRadi(i)*(depthRedBall(i)...
                *0.001933+0.02027);
            if  (mmradiusRedBalls(i) > (RadPingBall-5))  &&...
```

```
                    (mmradiusRedBalls(i) <= (RadPingBall + 5))...
                    && (size(redBallsRadi,1)>1)%Litt slingrings mon
                if i==2%som referanse til å finne orienteringen i X,Y planer
                    dist_red_Y=abs(redBallsCenter(i-1,2)-...
                        redBallsCenter(i,2))*(min(depthRedBall)...
                        *0.001933+0.02027); %Bruker den nærmeste kula
                    dist_red_X=abs(redBallsCenter(i-1,1)-...
                        redBallsCenter(i,1))*(min(depthRedBall)...
                        *0.001933+0.02027);
                end
            end
        end
    end

end
%%  Finne Ramme Fv
%Nedre høyre hjørne
p2vk=[(320-1)*796*0.001933;(240-480)*796*0.001933;796];
%nedre venstre hjørne i bilde., origo
p1vk=[(320-551)*830*0.001933;(240-411)*830*0.001933;830];
p3vk=[(320-551)*830*0.001933;(240-0)*830*0.001933;830]; %Øvre høyre hjørne

p2v=[sqrt(((551-1)*796*0.001933)^2+830^2);0;0]; p1v=[0;0;0];
p3v=[0;0;411*900*0.001933];

rv=p2v-p1v; tv=p3v-p1v; rn=p2vk-p1vk; tn=p3vk-p1vk;
c1v=rv/norm(rv); c2v=cross(rv,tv)/norm(cross(rv,tv));
c3v=cross(c1v,c2v); Rcv=[c1v c2v c3v];
c1n=rn/norm(rn); c2n=cross(rn,tn)/norm(cross(rn,tn));
c3n=cross(c1n,c2n); Rcn=[c1n c2n c3n]; Rnv=Rcn*Rcv';
pnv_manuel(:,1)=p1vk-Rnv*p1v; pnv_manuel(:,2)=p2vk-Rnv*p2v;
pnv_manuel(:,3)=p3vk-Rnv*p3v;

%Snitt av alle pbn vektorene
pnv_mean=(pnv_manuel(:,1)+pnv_manuel(:,2)+pnv_manuel(:,3))/3;
pnv_manuel_std=zeros(3,1);
for i=1:3
    %Finner standardavvik for pbn
pnv_manuel_std=pnv_manuel_std+(pnv_manuel(:,i)-pnv_mean).^2;
end
pnv_manuel_std=sqrt((1/2)*pnv_manuel_std);
%%
ub=p2b-p1b; vb=p3b-p1b; c1b=ub/norm(ub);
c2b=cross(ub,vb)/norm(cross(ub,vb));
c3b=cross(c1b,c2b); Rcb=[c1b c2b c3b];

p1n_manuel=[42*600*0.001933;-(30*600*0.001933);600];
p2n_manuel=[122*555*0.001933;-(31*555*0.001933);555];
p3n_manuel=[76*674*0.001933;-(50*674*0.001933);674];
un_manuel=p2n_manuel-p1n_manuel;
vn_manuel=p3n_manuel-p1n_manuel;

c1n_manuel=un_manuel/norm(un_manuel);
c2n_manuel=cross(un_manuel,vn_manuel)/norm(cross(ub,vb));
c3n_manuel=cross(c1n_manuel,c2n_manuel);
Rcn_manuel=[c1n_manuel c2n_manuel c3n_manuel];
Rbn_manuel=Rcn_manuel*Rcb';
pbn_manuel(:,1)=p1n_manuel-Rbn_manuel*p1b;
pbn_manuel(:,2)=p2n_manuel-Rbn_manuel*p2b;
pbn_manuel(:,3)=p3n_manuel-Rbn_manuel*p3b;

%Snitt av alle pbn vektorene
pbn_manuel_mean=(pbn_manuel(:,1)+pbn_manuel(:,2)+pbn_manuel(:,3))/3;
pbn_manuel_std=zeros(3,1);
for i=1:3
 %Finner standardavvik for pbn
pbn_manuel_std=pbn_manuel_std+(pbn_manuel(:,i)-pbn_manuel_mean).^2;
```

```
end
pbn_manuel_std=sqrt((1/2)*pbn_manuel_std);

p1n_auto=[(320-greenBallsCenter(2,1))*depthGreenBall(2)*0.001933;...
(240-greenBallsCenter(2,2))*depthGreenBall(2)*0.001933;depthGreenBall(2)];
p2n_auto=[(320-redBallsCenter(1,1))*depthRedBall(1)*0.001933;...
    (240-redBallsCenter(1,2))*depthRedBall(1)*0.001933;depthRedBall(1)];
p3n_auto=[(320-greenBallsCenter(1,1))*depthGreenBall(1)*0.001933;...
 (240-greenBallsCenter(1,2))*depthGreenBall(1)*0.001933;depthGreenBall(1)];
un_auto=p2n_auto-p1n_auto;
vn_auto=p3n_auto-p1n_auto;

c1n_auto=un_auto/norm(un_auto);
c2n_auto=cross(un_auto,vn_auto)/norm(cross(ub,vb));
c3n_auto=cross(c1n_auto,c2n_auto); Rcn_auto=[c1n_auto c2n_auto c3n_auto];

Rbn_auto=Rcn_auto*Rcb';
pbn_auto(:,1)=p1n_auto-Rbn_auto*p1b;
pbn_auto(:,2)=p2n_auto-Rbn_auto*p2b;
pbn_auto(:,3)=p3n_auto-Rbn_auto*p3b;

%Snitt av alle pbn vektorene
pbn_auto_mean=(pbn_auto(:,1)+pbn_auto(:,2)+pbn_auto(:,3))/3;
pbn_auto_std=zeros(3,1);
for i=1:3
 %Finner standardavvik for pbn
pbn_auto_std=pbn_auto_std+(pbn_auto(:,i)-pbn_auto_mean).^2;
end
pbn_auto_std=sqrt((1/2)*pbn_auto_std);

quadCorner=[p1b p3b p2b p4b];
%Plotter quad i n-ramma
for i=1:4
 quad(:,i)=(-Rnv'*pnv_mean)+Rnv'*(pbn_auto_mean+Rbn_auto*quadCorner(:,i));
end
[x,y,z] = sphere;
figure
r=0; g=1;
for i=1:size(quad,2)
    if i==3
        r=1; g=0;
    end
    surf(RadPingBall*x+quad(1,i),RadPingBall*y+quad(2,i),...
        RadPingBall*z+quad(3,i),'FaceColor',[r g 0],'EdgeColor','none')
    hold on
end
view(167,10); xax=640; yax=480; zax=max(max(double(depthFrame)));
axis([0 800 0 800 0 800]); xlabel('X-akse'); ylabel('Y-akse');
zlabel('Z-akse'); camlight left; lighting phong
%%
diffImage=double(diffImage);
%Ringer rundt markeringsballene
diffImage=insertObjectAnnotation(diffImage,'circle',...
    [greenBallsCenter greenBallsRadi],[1 2],'Color','green');
diffImage=insertObjectAnnotation(diffImage,'circle',...
    [redBallsCenter redBallsRadi],[1 2],'Color','red');

figure;
imshow(diffImage);
figure;
subplot(1,2,1);
imshow(binFrameRed);
subplot(1,2,2);
imshow(binFrameGreen);
```

Appendiks A-3: Orientering fra opptak

```
%############### Orientering av quadrokopter fra opptak #################

% Dette programmet stepper igjennom opptak tatt av et quadrokopter,
% detekterer QK, finner orienteringen og posisjon og markerer de
% detekterte markeringsballen og plotter posisjonen og orienteringen

%########################################################################
clc; clear all; close all;
%Laster opptakene og referansebildene
load('RGB_Opptak'); load('Depth_Opptak'); load('Kryss'); load('Trekant');
load('Likhetstegn'); template=kryss;
RadPingBall=20;% Radius til bordtennisball(mm)
redThresh = 0.24; % Threshold for rød farge
greenThresh = 0.05; % Threshold for grønn farge
blueThresh = 0.30; % Threshold for blå farge

%Vektorene til navigasjons ramma
p1n=[300;0;0];      %Pluss tegn
p2n=[0;-300;0];     %Likhets tegn
p3n=[0;0;300];      %Trekant tegn

%Vektorene til Quad i quad ramma
p1b=[50;0;60];     %Grønn kule. Kjøre retning
p2b=[-50;50;50];   %Blå kule. Venstre kule for kjøre retningen
p3b=[-50;-50;50];      %Rød kule. Høyre kule for kjøre retningen
quadCorner=[p1b p2b p3b];

%Setter opp objektdeteksjon for de binære bildene
obj_det = vision.BlobAnalysis('AreaOutputPort', false, ...
                              'CentroidOutputPort', true, ...
                              'BoundingBoxOutputPort', true', ...
                              'MinimumBlobArea', 200, ...
                              'MaximumBlobArea', 10000, ...
                              'MaximumCount', 1);
%Setter opp funksjonen for å markere detekterte objekter
farge_boks = vision.ShapeInserter('BorderColorSource', 'Input port', ...
                                  'Fill', true, ...
                                  'FillColorSource', 'Input port',...
                                  'Opacity', 0.4);

senter_tekst = vision.TextInserter('Text', '+    X:%4d, Y:%4d', ...
                                   'LocationSource', 'Input port', ...
                                   'Color', [1 1 0], ...
                                   'Font', 'Courier New', ...
                                   'FontSize', 14);
%Setter opp videospilleren
Video_ut = vision.VideoPlayer('Name', 'Quad Opptak', ...
                              'Position', [100 100 660 510]);
%Flipper opptakene da opptakene fra Kinect er speilvendt
RGB_Opptak=flipdim(RGB_Opptak,2);
Depth_Opptak=flipdim(Depth_Opptak,2);

 %Tar vare på forrige dybde måling, midlertidig verdi
 depthGreenBall_last= 6000; depthBlueBall_last= 6000;depthRedBall_last= 6000;

 figure(1)
% %
for i=1:size(RGB_Opptak,4)
        rgbFrame=RGB_Opptak(:,:,:,i);
        depthFrame=Depth_Opptak(:,:,i);
        RGB=imresize(rgbFrame(12:480-37,34:640-34,:),[480 640]);
        IR=[depthFrame(:,15:640) zeros(480,15)];
        RGB_gray=rgb2gray(RGB);
%
```

```
    if i==1% Finner navigasjons ramma
        for j=1:3
             if j==2
                template=likhetstegn;
            elseif j==3
                template=trekant;
            end
           %Leter etter symboler som markerer n-ramma
           cc = normxcorr2(template,RGB_gray);
           [max_cc, imax] = max(abs(cc(:)));
           [ypeak, xpeak] = ind2sub(size(cc),imax(1));
           corr_offset = [ (ypeak-size(template,1)) ...
               (xpeak-size(template,2))];

           depth=double(IR(corr_offset(1)+size(template,1),...
               corr_offset(2)+size(template,2)));
           %Finner vektorene som definerer n-ramma
           if j==1
               p1k=[(320-corr_offset(2))*depth*0.0017;...
                   (240-corr_offset(1))*depth*0.0017;depth];
           elseif j==2
               p2k=[(320-corr_offset(2))*depth*0.0017;...
                   (240-corr_offset(1))*depth*0.0017;depth];
           else
               p3k=[(320-corr_offset(2))*depth*0.0017;...
                   (240-corr_offset(1))*depth*0.0017;depth];
           end
        end
%% Finner retningkosinmatrisa fra Kinect ramma til
%Navigasjons ramma og vektoren,pkn, fra Kinect basen til Navigasjons basen
                rn=p2n-p1n; tn=p3n-p1n; rk=p2k-p1k; tk=p3k-p1k;
                c1n=rn/norm(rn);c2n=cross(rn,tn)/norm(cross(rn,tn));
                c3n=cross(c1n,c2n); Rcn=[c1n c2n c3n];
                c1k=rk/norm(rk); c2k=cross(rk,tk)/norm(cross(rk,tk));
                c3k=cross(c1k,c2k); Rck=[c1k c2k c3k];
                Rnk=Rck*Rcn';

                pkn(:,1)=p1k-Rnk*p1n; pkn(:,2)=p2k-Rnk*p2n; pkn(:,3)=p3k-Rnk*p3n;
                 %Snitt av alle pbn vektorene
                pkn_mean=(pkn(:,1)+pkn(:,2)+pkn(:,3))/3;
     end
    %Segmenterer ut markeringsballene
    diffFrameRed=imsubtract(RGB(:,:,1),RGB_gray);
    binFrameRed = im2bw(diffFrameRed, redThresh);

    diffFrameGreen=imsubtract(RGB(:,:,2),RGB_gray);
    binFrameGreen = im2bw(diffFrameGreen, greenThresh);

    diffFrameBlue=imsubtract(RGB(:,:,3),RGB_gray);
    binFrameBlue = im2bw(diffFrameBlue, blueThresh);

    %Finner posisjon og størrelse på objektene
    [redBallCenter, redArea]=step(obj_det,binFrameRed);
    [greenBallCenter,greenArea]=step(obj_det,binFrameGreen);
    [blueBallCenter,blueArea]=step(obj_det,binFrameBlue);

    %% Viser visuelt på video hvor markeringskulene er
    vidIn = step(farge_boks, RGB, redArea, uint8([255 0 0]));
    vidIn = step(farge_boks, vidIn, greenArea, uint8([0 255 0]));
    vidIn = step(farge_boks, vidIn, blueArea, uint8([0 0 255]));

    %Setter inn tekst på bilde hvilke koordinater markeringsballene har
    if ~isempty(redBallCenter)
        centXRed = uint16(redBallCenter(1,1)); centYRed = ...
            uint16(redBallCenter(1,2));
        vidIn = step(senter_tekst, vidIn, [centXRed centYRed], ...
            [centXRed-6 centYRed-9]);
```

```
    end
    if ~isempty(greenBallCenter)
        centXGreen = uint16(greenBallCenter(1,1)); centYGreen = ...
            uint16(greenBallCenter(1,2));
        vidIn = step(senter_tekst, vidIn, [centXGreen centYGreen], ...
            [centXGreen-6 centYGreen-9]);
    end

    if ~isempty(blueBallCenter)
        centXBlue = uint16(blueBallCenter(1,1)); centYBlue = ...
            uint16(blueBallCenter(1,2));
        vidIn = step(senter_tekst, vidIn, [centXBlue centYBlue],...
            [centXBlue-6 centYBlue-9]);
    end
%%  Finner retningkosinmatrisa fra Quad til Kinect
    if ~isempty(greenBallCenter) && ~isempty(blueBallCenter) &&...
            ~isempty(redBallCenter)

        depthGreenBall=double(IR(round(greenBallCenter(1,2)),...
            round(greenBallCenter(1,1))));
        depthBlueBall=double(IR(round(blueBallCenter(1,2)),...
            round(blueBallCenter(1,1))));
        depthRedBall=double(IR(round(redBallCenter(1,2)),...
            round(redBallCenter(1,1))));

        %Sjekker om det er rimelige dybdedata
        if (abs(depthRedBall-depthGreenBall) > 100 || abs(depthBlueBall...
                -depthGreenBall)> 100) && depthGreenBall_last ~= 0
            depthGreenBall=depthGreenBall_last;
        else
            depthGreenBall_last= depthGreenBall;
        end

        if (abs(depthGreenBall-depthRedBall) > 100 || abs(depthBlueBall...
                -depthRedBall) > 100) && depthRedBall_last ~=0
            depthRedBall=depthRedBall_last;
        else
            depthBlueBall_last = depthBlueBall;
        end

        if (abs(depthGreenBall-depthBlueBall) > 100 || abs(depthRedBall...
                -depthBlueBall) > 100) && depthBlueBall_last ~=0
            depthBlueBall=depthBlueBall_last;
        else
           depthRedBall_last= depthRedBall;
        end
        greenArea=double(greenArea); blueArea=double(blueArea);
        redArea=double(redArea);

        ub=p2b-p1b;vb=p3b-p1b; c1b=ub/norm(ub);
        c2b=cross(ub,vb)/norm(cross(ub,vb));
        c3b=cross(c1b,c2b); Rcb=[c1b c2b c3b];
        %Finner vektorene til markeringsballene sett ifra Kinect
        q1k=[(320-greenBallCenter(:,1))*(RadPingBall/(greenArea(3)/2));...
            (240-greenBallCenter(:,2))*(RadPingBall/(greenArea(4)/2));...
            depthGreenBall];
        q2k=[(320-blueBallCenter(:,1))*(RadPingBall/(blueArea(3)/2));...
            (240-blueBallCenter(:,2))*(RadPingBall/(blueArea(4)/2));...
            depthBlueBall];
        q3k=[(320-redBallCenter(:,1))*(RadPingBall/(redArea(3)/2));...
      (240-redBallCenter(:,2))*(RadPingBall/(redArea(4)/2));depthRedBall];

        uk=q2k-q1k;vk=q3k-q1k; c1k=uk/norm(uk);c2k=cross(uk,vk)/norm(cross(uk,vk));
        c3k=cross(c1k,c2k); Rck=[c1k c2k c3k]; Rbk=Rck*Rcb';
        pbk(:,1)=q1k-Rbk*p1b; pbk(:,2)=q2k-Rbk*p2b; pbk(:,3)=q3k-Rbk*p3b;
        %Snitt av alle pbn vektorene
        pbk_mean=(pbk(:,1)+pbk(:,2)+pbk(:,3))/3;
```

```
%%    Simulering av Quaddrocopteret
        clf
        for j=1:3
            quad(:,j)=(-Rnk'*pkn_mean)+Rnk'*(pbk_mean+Rbk*quadCorner(:,j));
        end
        [x,y,z] = sphere;
        r=0; g=1; b=0;
        for j=1:size(quad,2)
             if j==2
                r=0;g=0; b=1;
             elseif j==3
                r=1; g=0; b=0;
             end
 %Plotter Quadrokopter i n-ramma
         subplot(1,2,1)
         surf(RadPingBall*x+quad(1,j),RadPingBall*y+quad(2,j),RadPingBall...
             *z+quad(3,j),'FaceColor',[r g b],'EdgeColor','none')
         hold on
        end
        hold off
        view(10,18); axis([0 1500 -1500 0 0 1500]);xlabel('X-akse');
        ylabel('Y-akse'); zlabel('Z-akse'); camlight left; lighting phong
        yaw_set=Rnk'*Rbk*[100;0;0];
        yaw_angle=acos(yaw_set(1)/sqrt(yaw_set(2)^2+yaw_set(1)^2));
        if yaw_set(2)<0
            yaw_angle=pi+(pi-yaw_angle);
        end
        subplot(1,2,2); pol=polar([0 yaw_angle],[0 1]); set(pol,'LineWidth',5)
        title('Yaw-vinkel i forhold til n-ramma')
        drawnow
    else
        clf
    end
%%
step(Video_ut, vidIn);
pause(0.033);
end
```

Appendiks A-4: Posisjons og nøyaktighetsmåling

```
%####################   Posisjon og nøyaktighetsmåling #################
% Dette programmet logger data av et QK som plasseres i n-ramma
% hvor det blir posisjonsnøyaktigheten og støy blir målt
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
clear all; close all; imaqreset; clc;
load('Kryss'); load('Trekant'); load('Likhetstegn');
template=kryss;
RadPingBall=20;             %Radius til bordtennisball

redThresh = 0.24;           % Threshold for rød farge
greenThresh = 0.07;         % Threshold for grønn farge
blueThresh = 0.25;          % Threshold for blå farge

p1n=[300;0;0];              %Pluss tegn  Vektorene til navigasjons ramma
p2n=[0;-300;0];             %Likhets tegn
p3n=[0;0;300];              %Trekant tegn

                            %Vektorene til Quad i quad ramma
p1b=[120;0;30];             %Grønn kule. Kjøre retning
p2b=[-60;170;30];           %Blå kule. Venstre kule for kjøre retningen
p3b=[-60;-170;30];          %Rød kule. Høyre kule for kjøre retningen

TP=[600;-300;0];             %Et kryss markert i navigasjons ramma

quadCorner=[p1b p2b p3b];
```

```
%Setter opp fargekameraet
vidDevice = imaq.VideoDevice('kinect', 1, 'YUV_640x480', ...
                    'ROI', [1 1 640 480], ...
                    'ReturnedColorSpace', 'rgb');

%Setter opp til dybdeopptak
depthInput=videoinput('kinect',2);
src = getselectedsource(depthInput);
triggerconfig(depthInput, 'manual');
depthInput.FramesPerTrigger=1;
src.DepthMode = 'Near';
depthInput.TriggerRepeat = Inf;
start(depthInput);                      %starte stream

%Setter opp objektderektsjon
objekt_deteksjon = vision.BlobAnalysis('AreaOutputPort', false, ...
                                'CentroidOutputPort', true, ...
                                'BoundingBoxOutputPort', true', ...
                                'MinimumBlobArea', 40, ...
                                'MaximumBlobArea', 5000, ...
                                'MaximumCount', 1);

fargeboks = vision.ShapeInserter('BorderColorSource', 'Input port', ...
                                      'Fill', true, ...
                                    'FillColorSource', 'Input port', ...
                                      'Opacity', 0.4);


%Setter inn tekst i videostreamen, for å vise posisjon pikselvis
senter_tekst = vision.TextInserter('Text', '+    X:%4d, Y:%4d', ...
                                   'LocationSource', 'Input port', ...
                                   'Color', [0 0 0], ...
                                   'Font', 'Courier New', ...
                                   'FontSize', 14);

 video_spiller = vision.VideoPlayer('Name', 'Video ut', ...
                                 'Position', [100 100 660 510]);
%%
%Tar vare på forrige dybde måling
 depthGreenBall_last= 6000;
 depthBlueBall_last= 6000;
 depthRedBall_last= 6000;
position_check=0;
                            %% Prosesloop
for i=1:10                        %Tar 10 bilder for å være på sikre siden
    RGB = step(vidDevice);
    trigger(depthInput)
    depthFrame=getdata(depthInput);
end
    RGB = flipdim(RGB,2);
    depthFrame=flipdim(depthFrame,2);

    RGB=imresize(RGB(12:480-37,34:640-34,:),[480 640]);
    IR=[depthFrame(:,15:640) zeros(480,15)];

    RGB_gray=rgb2gray(RGB);

        for j=1:3
            if j==2
                template=likhetstegn;
            elseif j==3
                template=trekant;
            end
           cc = normxcorr2(template,RGB_gray);
           [max_cc, imax] = max(abs(cc(:)));
```

```
            [ypeak, xpeak] = ind2sub(size(cc),imax(1));
            corr_offset = [ (ypeak-size(template,1)) ...
                (xpeak-size(template,2)) ];

            depth=double(IR(corr_offset(1)+size(template,1),...
                corr_offset(2)+size(template,2)));
            if j==1
               p1k=[(320-(corr_offset(2)+size(template,2)))*depth*0.0017;...
               (240-(corr_offset(1)+size(template,1)))*depth*0.0017;depth];
            elseif j==2
                p2k=[(320-corr_offset(2))*depth*0.0017;...
                (240-(corr_offset(1)+size(template,1)))*depth*0.0017;depth];
            else
                p3k=[(320-corr_offset(2))*depth*0.0017;...
                    (240-corr_offset(1))*depth*0.0017;depth];
            end
        end

    %%Finner retningkosinmatrisa og vektoren mellom basene  til
    %%navigasjonsramma
    rn=p2n-p1n; tn=p3n-p1n; rk=p2k-p1k; tk=p3k-p1k;
    c1n=rn/norm(rn);c2n=cross(rn,tn)/norm(cross(rn,tn));c3n=cross(c1n,c2n);
    Rcn=[c1n c2n c3n];

    c1k=rk/norm(rk);c2k=cross(rk,tk)/norm(cross(rk,tk));c3k=cross(c1k,c2k);
    Rck=[c1k c2k c3k];
    Rnk=Rck*Rcn';
    pkn(:,1)=p1k-Rnk*p1n; pkn(:,2)=p2k-Rnk*p2n; pkn(:,3)=p3k-Rnk*p3n;
    %Snitt av alle pbn vektorene
    pkn_mean=(pkn(:,1)+pkn(:,2)+pkn(:,3))/3;

for u=1:200

    RGB = step(vidDevice);
    RGB = flipdim(RGB,2);
    trigger(depthInput)
    depthFrame=getdata(depthInput);
    depthFrame=flipdim(depthFrame,2);
    RGB=imresize(RGB(15:480-37,34:640-34,:),[480 640]);
    RGB_gray=rgb2gray(RGB);
    IR=[depthFrame(:,5:640) zeros(480,5)];
 %%                          Finne Rød Kule
     diffFrameRed=imsubtract(RGB(:,:,1),RGB_gray);
     binFrameRed = im2bw(diffFrameRed, redThresh);
    [redBallCenter, redArea]=step(objekt_deteksjon,binFrameRed);
 %%#####################################################################
 %%                        Finne Grønn Kule
    diffFrameGreen = imsubtract(RGB(:,:,2), RGB_gray);
    binFrameGreen = im2bw(diffFrameGreen, greenThresh);
   [greenBallCenter,greenArea]=step(objekt_deteksjon,binFrameGreen);
%%%###################################################################
%%                Finne blå kule
    diffFrameBlue = imsubtract(RGB(:,:,3), RGB_gray);
    binFrameBlue = im2bw(diffFrameBlue, blueThresh);
    [blueBallCenter,blueArea]=step(objekt_deteksjon,binFrameBlue);
%####################################################################
%%
     greenArea=double(greenArea);
     blueArea=double(blueArea);
     redArea=double(redArea);
% Finner orientering til Quad
     if ~isempty(greenBallCenter) && ~isempty(blueBallCenter) && ...
             ~isempty(redBallCenter)
        depthGreenBall(u)=double(IR(round(greenBallCenter(1,2)),...
            round(greenBallCenter(1,1))));
        depthBlueBall(u)=double(IR(round(blueBallCenter(1,2)),...
            round(blueBallCenter(1,1))));
```

```
        depthRedBall(u)=double(IR(round(redBallCenter(1,2)),...
            round(redBallCenter(1,1))));

        ub=p2b-p1b; vb=p3b-p1b; c1b=ub/norm(ub);
        c2b=cross(ub,vb)/norm(cross(ub,vb)); c3b=cross(c1b,c2b);
        Rcb=[c1b c2b c3b];

        pg1k=[(320-greenBallCenter(:,1))*depthGreenBall(u)*0.0017;...
    (240-greenBallCenter(:,2))*depthGreenBall(u)*0.0017;depthGreenBall(u)];
        pb2k=[(320-blueBallCenter(:,1))*depthBlueBall(u)*0.0017;...
       (240-blueBallCenter(:,2))*depthBlueBall(u)*0.0017;depthBlueBall(u)];
        pr3k=[(320-redBallCenter(:,1))*depthRedBall(u)*0.0017;...
          (240-redBallCenter(:,2))*depthRedBall(u)*0.0017;depthRedBall(u)];

        uk=pb2k-pg1k; vk=pr3k-pg1k; c1k=uk/norm(uk);
        c2k=cross(uk,vk)/norm(cross(uk,vk)); c3k=cross(c1k,c2k);
        Rck=[c1k c2k c3k];
        Rbk=Rck*Rcb';

      pbk(:,1)=pg1k-Rbk*p1b; pbk(:,2)=pb2k-Rbk*p2b; pbk(:,3)=pr3k-Rbk*p3b;
        %Snitt av alle pbn vektorene
       pbk_mean=(pbk(:,1)+pbk(:,2)+pbk(:,3))/3;

        if position_check ~=2
        position_check=1;
        end
%Finner quadrokopterets posisjon i navigasjons ramma
        for j=1:3
            quad(:,j)=(-Rnk'*pkn_mean)+Rnk'*(pbk_mean+Rbk*quadCorner(:,j));
            if quad(3,j)<=0
                quad(3,j)=0;
            end
        end
    end
%%
    %Setter inn fargede bokser for å vise markeringsballene
    vidIn = step(fargeboks, RGB, redArea, single([255 0 0]));
    vidIn = step(fargeboks, vidIn, greenArea, single([0 255 0]));
    vidIn = step(fargeboks, vidIn, blueArea, single([0 0 255]));

    if ~isempty(redBallCenter)
        centXRed = uint16(redBallCenter(1,1)); centYRed = uint16...
            (redBallCenter(1,2));
        vidIn = step(senter_tekst, vidIn, [centXRed centYRed], ...
            [centXRed-6 centYRed-9]);
    end
    if ~isempty(greenBallCenter)
        centXGreen = uint16(greenBallCenter(1,1)); centYGreen =...
            uint16(greenBallCenter(1,2));
        vidIn = step(senter_tekst, vidIn, [centXGreen centYGreen],...
            [centXGreen-6 centYGreen-9]);
    end
    if ~isempty(blueBallCenter)
        centXBlue = uint16(blueBallCenter(1,1)); centYBlue =...
            uint16(blueBallCenter(1,2));
        vidIn = step(senter_tekst, vidIn, [centXBlue centYBlue],...
            [centXBlue-6 centYBlue-9]);
    end
       step(video_spiller, vidIn);
  if ~isempty(greenBallCenter) && ~isempty(blueBallCenter) &&...
          ~isempty(redBallCenter) && ~isnan(pbk_mean(1))

         quad_center=(-Rnk'*pkn_mean)+Rnk'*pbk_mean;
    if position_check==1
        start_center=quad_center;
        desired_position=[quad_center(1:2);quad_center(3)];
```

```
        position_check=2;
    end
    position_error(:,u)=desired_position-quad_center;

    yaw_set=Rnk'*Rbk*[100;0;0];
    yaw_angle_measure=acos(yaw_set(1)/sqrt(yaw_set(2)^2+yaw_set(1)^2));

    if yaw_set(2)<0
        yaw_angle_measure=pi+(pi-yaw_angle_measure);
    end
    yaw_angle(u)=yaw_angle_measure;
  end
end
%%
figure(1)
plot(1:u,position_error(1,:),1:u,position_error(2,:),...
    1:u,position_error(3,:));
legend('Endring i X-akse','Endring i Y-akse','Endring i Z-akse');
xlabel('Tidsforløp i antall sample');
ylabel('Avvik i mm');
title('Avvik i posisjonsmåling over tid.(Stasjonært)')
figure(2)
j=1;
x=1;
for i=1:200
    if j==10
        snitt_xfeil(x)=mean(position_error(1,i-9:i));
        snitt_yfeil(x)=mean(position_error(2,i-9:i));
        snitt_zfeil(x)=mean(position_error(3,i-9:i));

        std_xfeil(x)=std(position_error(1,i-9:i));
        std_yfeil(x)=std(position_error(2,i-9:i));
        std_zfeil(x)=std(position_error(3,i-9:i));

        snitt_dfeilGball(x)=mean(depthGreenBall(i-9:i));
        snitt_dfeilBball(x)=mean(depthBlueBall(i-9:i));
        snitt_dfeilRball(x)=mean(depthRedBall(i-9:i));

        std_dfeilGball(x)=std(depthGreenBall(i-9:i));
        std_dfeilBball(x)=std(depthBlueBall(i-9:i));
        std_dfeilRball(x)=std(depthRedBall(i-9:i));
        x=x+1;
        j=0;
    end
    j=j+1;
end
subplot(1,2,1)
plot(1:20,snitt_xfeil(:),1:20,snitt_yfeil(:),1:20,snitt_zfeil(:));
legend('X-akse','Y-akse','Z-akse','Location','Best');
xlabel('Tidsforløp i antall sample');
ylabel('Avvik i mm');
title('Snitt av avvik fra setpunkt. 10 og 10 målinger')
subplot(1,2,2)
plot(1:20,std_xfeil(:),1:20,std_yfeil(:),1:20,std_zfeil(:));
legend('X-akse','Y-akse','Z-akse','Location','Best');
xlabel('Tidsforløp i antall sample');
ylabel('Avvik i mm');
title('Standardavvik for posisjonsavvik fra setpunkt. 10 og 10 målinger')

figure(3)
plot(1:u,depthGreenBall(:),'g',1:u,depthBlueBall(:),'b',1:u,...
    depthRedBall(:),'r');
legend('Dybde grønn ball','Dybde blå ball','Dybde rød ball');
xlabel('Tidsforløp i antall sample')
ylabel('Målte dybder til markerings baller')
title('Variasjon i dybdemålinger til stasjonar Quadrokopter')
```

```
figure(4)
subplot(1,2,1)
plot(1:20,snitt_dfeilBball(:),'b',1:20,snitt_dfeilGball,'g',1:20,...
    snitt_dfeilRball,'r');
legend('Blå ball','Grønn ball','Rød ball','Location','Best');
xlabel('Tidsforløp i antall sample');
ylabel('Avvik i mm');
title('Snitt av 10 og 10 dybde målinger')
subplot(1,2,2)
plot(1:20,std_dfeilBball(:),'b',1:20,std_dfeilGball(:),'g',1:20,...
    std_dfeilRball(:),'r');
legend('Blå ball','Grønn ball','Rød ball','Location','Best');
xlabel('Tidsforløp i antall sample');
ylabel('Avvik i mm');
title('Standardavvik for 10 og 10 dybde målinger')
%Plotter posisjonen til Quadrokopteret i navigasjons ramma
[x,y,z] = sphere;
        r=0; g=1; b=0;
        for j=1:size(quad,2)
             if j==2
                r=0; g=0; b=1;
             elseif j==3
                r=1; g=0; b=0;
             end
         figure(6)
         surf(RadPingBall*x+quad(1,j),RadPingBall*y+quad(2,j),...
            RadPingBall*z+quad(3,j),'FaceColor',[r g b],'EdgeColor','none')
         hold on
        end
        view(10,18)
        axis([0 2000 -2000 0 0 2000]);
        xlabel('X-akse');
        ylabel('Y-akse');
        zlabel('Z-akse');
        camlight left; lighting phong

        disp('Ønsket posisjon i mm:[600;-300;0]')
        disp('Målt(mm):');disp(quad_center)
        disp('Differanse(mm):');disp(abs(quad_center-TP))

        figure(8)
        subplot(2,2,1)
        yaw_std=std(yaw_angle);
        plot(1:u,yaw_angle(:)*180/pi);
        title('Yaw vinkel i forhold til n-ramma')

        subplot(2,2,2)
        pol=polar([0 mean(yaw_angle)],[0 1],'r');
        set(pol,'LineWidth',3)
        title('Yaw snittmåling')

        subplot(2,2,4)
        pol=polar([0 yaw_std],[0 1],'b');
        set(pol,'LineWidth',3)
        title('std Yaw')

        annotation('textbox',[0.15 0.15 0.3 0.15],'string',...
            ['Standardavvik:' num2str(yaw_std*180/pi) char(176)])
    %%
release(video_spiller);
release(vidDevice);
```

Appendiks A-5: Autopilot

```
%########################## Kinect Autopilot ##############################
% Dette programmet finner QK, setter setpunkt til 200 mm over
% startsposisjon, finner avvik og sender styringsignal serielt til
% mikrokontrolleren.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
close all
imaqreset
delete(instrfindall);
clc
sprek_pc=0;     %Brukes en rask PC, kan denne settes til 1. Default 0
stop=0;

%Setter opp COM port
s=serial('COM6');
fopen(s);
flushinput(s);

load('Kryss');
load('Trekant');
load('Likhetstegn');

template=kryss;

RadPingBall=20;% Radius til bordtennisball

redThresh = 0.24; % Threshold for red detection
greenThresh = 0.07; % Threshold for green detection
blueThresh = 0.25; % Threshold for blue detection

p1n=[300;0;0];     %Pluss tegn  Vektorene til navigasjons ramma
p2n=[0;-300;0];    %Likhets tegn
p3n=[0;0;300];     %Trekant tegn
%Vektorene til Quad i quad ramma
p1b=[120;0;30];    %Grønn kule. Kjøre retning
p2b=[-60;170;30];   %Blå kule. Venstre kule for kjøre retningen
p3b=[-60;-170;30];     %Rød kule. Høyre kule for kjøre retningen
quadCorner=[p1b p2b p3b];

% PID Parametre
throttle_I=45;
tau=[];          %Tidsparameter til Integral ledd
dt=0.2060;       %Step intervall
throttle=0;
yaw_error=0;
yaw_I=20;
pt=0;           %Pitch P timer
pitch_I=0;

vidDevice = imaq.VideoDevice('kinect', 1, 'YUV_640x480', ...
                    'ROI', [1 1 640 480], ...
                    'ReturnedColorSpace', 'rgb');
vidInfo = imaqhwinfo(vidDevice);

depthInput=videoinput('kinect',2);
src = getselectedsource(depthInput);
triggerconfig(depthInput, 'manual');
depthInput.FramesPerTrigger=1;
src.DepthMode = 'Near';
depthInput.TriggerRepeat = Inf;
start(depthInput);                      %starte stream

obj_det = vision.BlobAnalysis('AreaOutputPort', false, ...
                              'CentroidOutputPort', true, ...
```

```
                                'BoundingBoxOutputPort', true', ...
                                'MinimumBlobArea', 50, ...
                                'MaximumBlobArea', 5000, ...
                                'MaximumCount', 1);
farge_boks = vision.ShapeInserter('BorderColorSource', 'Input port', ...
                                      'Fill', true, ...
                                    'FillColorSource', 'Input port', ...
                                      'Opacity', 0.4);
htextinsCent = vision.TextInserter('Text', '+    X:%4d, Y:%4d', ...
                                   'LocationSource', 'Input port', ...
                                   'Color', [1 1 0], ...
                                   'Font', 'Courier New', ...
                                   'FontSize', 14);
if sprek_pc==1;
 video = vision.VideoPlayer('Name', 'Autopilot', ...
                                 'Position', [100 100 660 510]);
end
%%
%Tar vare på forrige dybde måling
depthGreenBall_last= 6000;
depthBlueBall_last= 6000;
depthRedBall_last= 6000;
position_check=1;
pos_lock=0;

for i=1:10                      %Tar 10 bilder for å være på sikre siden
    RGB = step(vidDevice);
    trigger(depthInput)
    depthFrame=getdata(depthInput);

end
    RGB = flipdim(RGB,2);
    depthFrame=flipdim(depthFrame,2);
    RGB=imresize(RGB(12:480-37,34:640-34,:),[480 640]);
    IR=[depthFrame(:,15:640) zeros(480,15)];
    RGB_gray=rgb2gray(RGB);
        for j=1:3
            if j==2
                template=likhetstegn;
            elseif j==3
                template=trekant;
            end
           cc = normxcorr2(template,RGB_gray);
           [max_cc, imax] = max(abs(cc(:)));
           [ypeak, xpeak] = ind2sub(size(cc),imax(1));
           corr_offset = [ (ypeak-size(template,1)) ...
               (xpeak-size(template,2)) ];
           depth=double(IR(corr_offset(1)+size(template,1),...
               corr_offset(2)+size(template,2)));
           if j==1
               p1k=[(320-(corr_offset(2)+size(template,2)))...
                   *depth*0.0017;(240-(corr_offset(1)+...
                   size(template,1)))*depth*0.0017;depth];
           elseif j==2
               p2k=[(320-corr_offset(2))*depth*0.0017;...
              (240-(corr_offset(1)+size(template,1)))*depth*0.0017;depth];
           else
               p3k=[(320-corr_offset(2))*depth*0.0017;...
                   (240-corr_offset(1))*depth*0.0017;depth];
           end
        end
    %%Finner retningkosinmatrisa og vektoren mellom basene  til
    %%navigasjonsramma


  rn=p2n-p1n; tn=p3n-p1n; rk=p2k-p1k; tk=p3k-p1k;
  c1n=rn/norm(rn); c2n=cross(rn,tn)/norm(cross(rn,tn)); c3n=cross(c1n,c2n);
```

```
  Rcn=[c1n c2n c3n];
  c1k=rk/norm(rk); c2k=cross(rk,tk)/norm(cross(rk,tk)); c3k=cross(c1k,c2k);
  Rck=[c1k c2k c3k];
  Rnk=Rck*Rcn';
  pkn(:,1)=p1k-Rnk*p1n; pkn(:,2)=p2k-Rnk*p2n; pkn(:,3)=p3k-Rnk*p3n;
  pkn_mean=(pkn(:,1)+pkn(:,2)+pkn(:,3))/3; %Snitt av alle pbn vektorene

  u=1;
while(stop==0)
    RGB = step(vidDevice);
    RGB = flipdim(RGB,2);
    trigger(depthInput)
    depthFrame=getdata(depthInput);
    depthFrame=flipdim(depthFrame,2);

    %Fra manuelle kalibreringen
    RGB=imresize(RGB(15:480-37,34:640-34,:),[480 640]);
    RGB_gray=rgb2gray(RGB);
    IR=[depthFrame(:,5:640) zeros(480,5)];
 %%                          Finne Rød Kule
    diffFrameRed=imsubtract(RGB(:,:,1),RGB_gray);
    binFrameRed = im2bw(diffFrameRed, redThresh);
    [redBallCenter, redArea]=step(obj_det,binFrameRed);
 %%#####################################################################
 %%                       Finne Grønn Kule
    diffFrameGreen = imsubtract(RGB(:,:,2), RGB_gray);
    binFrameGreen = im2bw(diffFrameGreen, greenThresh);
   [greenBallCenter,greenArea]=step(obj_det,binFrameGreen);
%%%###################################################################
%%               Finne blå kule
    diffFrameBlue = imsubtract(RGB(:,:,3), RGB_gray);
    binFrameBlue = im2bw(diffFrameBlue, blueThresh);
    [blueBallCenter,blueArea]=step(obj_det,binFrameBlue);
%###################################################################
     greenArea=double(greenArea);
     blueArea=double(blueArea);
     redArea=double(redArea);
% Finner orientering til Quad
     if ~isempty(greenBallCenter) && ~isempty(blueBallCenter) &&...
             ~isempty(redBallCenter)
        pos_lock=1;
        depthGreenBall=double(IR(round(greenBallCenter(1,2)),...
            round(greenBallCenter(1,1))));
        depthBlueBall=double(IR(round(blueBallCenter(1,2)),...
            round(blueBallCenter(1,1))));
        depthRedBall=double(IR(round(redBallCenter(1,2)),...
            round(redBallCenter(1,1))));

        if (abs(depthRedBall-depthGreenBall) > 300 ||...
                abs(depthBlueBall-depthGreenBall)> 300) ...
                && depthGreenBall_last ~= 0
            depthGreenBall=depthGreenBall_last;
        else
            depthGreenBall_last= depthGreenBall;
        end

        if (abs(depthGreenBall-depthRedBall) > 300 ||...
           abs(depthBlueBall-depthRedBall) > 300) && depthRedBall_last ~=0
            depthRedBall=depthRedBall_last;
        else
            depthBlueBall_last = depthBlueBall;
        end

        if (abs(depthGreenBall-depthBlueBall) > 300 ||...
                abs(depthRedBall-depthBlueBall) > 300)...
                && depthBlueBall_last ~=0
            depthBlueBall=depthBlueBall_last;
```

```
        else
           depthRedBall_last= depthRedBall;
        end
  ub=p2b-p1b; vb=p3b-p1b; c1b=ub/norm(ub);
  c2b=cross(ub,vb)/norm(cross(ub,vb)); c3b=cross(c1b,c2b);
  Rcb=[c1b c2b c3b];

  pg1k=[(320-greenBallCenter(:,1))*depthGreenBall*0.0017;...
      (240-greenBallCenter(:,2))*depthGreenBall*0.0017;depthGreenBall];
  pb2k=[(320-blueBallCenter(:,1))*depthBlueBall*0.0017;...
      (240-blueBallCenter(:,2))*depthBlueBall*0.0017;depthBlueBall];
  pr3k=[(320-redBallCenter(:,1))*depthRedBall*0.0017;...
      (240-redBallCenter(:,2))*depthRedBall*0.0017;depthRedBall];

 uk=pb2k-pg1k;vk=pr3k-pg1k; c1k=uk/norm(uk);
 c2k=cross(uk,vk)/norm(cross(uk,vk)); c3k=cross(c1k,c2k);
 Rck=[c1k c2k c3k];
 Rbk=Rck*Rcb';

 pbk(:,1)=pg1k-Rbk*p1b; pbk(:,2)=pb2k-Rbk*p2b;pbk(:,3)=pr3k-Rbk*p3b;
 pbk_mean=(pbk(:,1)+pbk(:,2)+pbk(:,3))/3;%Snitt av alle pbn vektorene
    end
%%
if sprek_pc==1
    vidIn = step(farge_boks, RGB, redArea, single([255 0 0]));
    vidIn = step(farge_boks, vidIn, greenArea, single([0 255 0]));
    vidIn = step(farge_boks, vidIn, blueArea, single([0 0 255]));
    if ~isempty(redBallCenter)
        centXRed = uint16(redBallCenter(1,1));...
            centYRed = uint16(redBallCenter(1,2));
        vidIn = step(htextinsCent, vidIn, ...
            [centXRed centYRed], [centXRed-6 centYRed-9]);
    end
    if ~isempty(greenBallCenter)
        centXGreen = uint16(greenBallCenter(1,1)); ...
            centYGreen = uint16(greenBallCenter(1,2));
        vidIn = step(htextinsCent, vidIn, ...
            [centXGreen centYGreen], [centXGreen-6 centYGreen-9]);
    end
    if ~isempty(blueBallCenter)
        centXBlue = uint16(blueBallCenter(1,1));...
            centYBlue = uint16(blueBallCenter(1,2));
        vidIn = step(htextinsCent, vidIn, ...
            [centXBlue centYBlue], [centXBlue-6 centYBlue-9]);
    end
 step(video, vidIn);
end

%     %%  Sende styrings signal til fjernkontroll
  if ~isempty(greenBallCenter) && ~isempty(blueBallCenter) ...
          && ~isempty(redBallCenter) && ~isnan(pbk_mean(1))

         quad_center=(-Rnk'*pkn_mean)+Rnk'*pbk_mean;

%######################### Setter settpunkt ##############################

% Første setpunkt settes til 200 mm ovenfor startposisjon
switch position_check
    case 1
        if pos_lock==1
            start_center=quad_center;
            desired_position=[quad_center(1:2);quad_center(3)+200];
            yaw_set=Rnk'*Rbk*[100;0;0];
            yaw_angle_set=acos(yaw_set(1)/sqrt(yaw_set(2)^2+yaw_set(1)^2));

            yaw_angle_measure_last=yaw_angle_set;
```

```
            position_check=2;
        end
    case 2
        if position_error(3)<=0
           yaw_angle_set=yaw_angle_set-pi; position_check=3;
        end

    case 3
        if yaw_error==0
           desired_position(3)=desired_position(3)-200; position_check=4;
        end
    case 4
        if position_error>=0
        stop=1;
        end
    otherwise
end

  position_error=desired_position-quad_center;
    yaw_set=Rnk'*Rbk*[100;0;0];
    yaw_angle_measure=acos(yaw_set(1)/sqrt(yaw_set(2)^2+yaw_set(1)^2));

    if yaw_set(2)<0
        yaw_angle_measure=pi+(pi-yaw_angle_measure);
    end
    yaw_error=yaw_angle_set-yaw_angle_measure;

        %Forsterkning
    Kpt=0.05;% Gasspådrag P
    Kpti=0.015;% Gasspådrag I

    Kpy=4;            % Yaw P
    Kpyi=0;%0.2;      % Yaw I

    Kpp=0.15;          %Pitch P
    Kppi=0.002;        %Pitch I
    Kpr=10;

    if ~isempty(tau)
        tau=toc;
    else
        tau=0;
    end
    tic;
    %###################### Throttle Gain ###################################
    %Throttle Gain P-ledd
    throttle_P=(Kpt*position_error(3));

    %Throttle Gain I-ledd
    throttle_I=throttle_I+Kpti*position_error(3)*(tau/dt);
    throttle_I(throttle_I>100)=100;
    throttle_I(throttle_I<0)=0;
    tau=1;

    throttle=throttle_P+throttle_I;     %Throttle ut
    throttle(throttle>100)=100;

    posisjon_feil(u)=position_error(3);
    paadrag_P(u)=throttle_P;
    paadrag_I(u)=throttle_I;
    %###################### YAW Gain ########################################
    yaw_P=83+Kpy*yaw_error;
    yaw_P(yaw_P>93)=93;
    yaw_P(yaw_P<73)=73;

    yaw_I=yaw_I+Kpyi*yaw_error*(tau/dt);
```

```
    yaw_I(yaw_I>25)=25; yaw_I(yaw_I<-25)=-25;
    yaw=yaw_P+yaw_I;

 %###################### Pitch Gain ###################################
    %Pitch Gain
    pitch_P=91-(Kpp*position_error(1));
    pitch_P(pitch_P>100)=100;
    pitch_P(pitch_P<75)=75;

    pitch_I=pitch_I+Kppi*position_error(1)*(tau/dt);
    pitch_I(pitch_I>5)=5;
    pitch_I(pitch_I<-5)=-3;

    if pt<=3
        pitch=pitch_P-pitch_I;
        pt=pt+1;
    else
        pitch=91-pitch_I;
        pt=0;
    end

 %#####################################################################
    %Roll Gain
    roll=50-(Kpr*position_error(2));
    roll(roll>100)=100;

  else
      yaw=83;
      pitch=91;
      throttle=throttle-0.01;
  end
        %Skriver til mikrokontroller
        if s.BytesAvailable ==0
              fwrite(s,uint8([240 throttle yaw pitch roll 250]));
          else
             flushinput(s);
        end
u=u+1;
end
%%
plot(1:u-1,paadrag_P(1:u-1),1:u-1,...
    paadrag_I(1:u-1),1:u-1,posisjon_feil(1:u-1));
if sprek_pc==1
    release(video);
end
release(vidDevice);
clear all;
clc;
```

Appendiks B: C-program Seriell-analog ut

```
/* PWM modulasjon for fjernkontroll
 Dette programmet venter på serielldata fra Matlab, for deretter å sette
 modulere PWM signaler basert på disse dataene til fjernkontrollen
 */
//Pinneutgangene for PWM moduleringen
int ledPin=13;
int pitch=3;
int roll=5;
int throttle=6;
int yaw=9;

int throttle_gain=6;
int yaw_gain=83;
```

```
int pitch_gain=91;
int roll_gain=77;

int throttle_gain_last=6;
int yaw_gain_last=83;
int pitch_gain_last=91;
int roll_gain_last=77;

int power=255*0.6;
int x[4];
int y;

void setup() {
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
  pinMode(throttle,OUTPUT);
  pinMode(yaw,OUTPUT);
  pinMode(pitch,OUTPUT);
  pinMode(roll,OUTPUT);

  Serial.begin(9600);

  analogWrite(yaw,yaw_gain);
  analogWrite(pitch,pitch_gain);
  analogWrite(roll, roll_gain);

  //En sekvens for å binde kontrollen med QK
  analogWrite(throttle,throttle_gain);
  delay(500);
  analogWrite(throttle,166);
  delay(500);
}

void loop() {

power=analogRead(A0)*0.249;

  if (Serial.available() >= 6) {
      y=Serial.read();
      if (y==240){
        x[0] = Serial.read();
        x[1] = Serial.read();
        x[2] = Serial.read();
        x[3] = Serial.read();
        y=Serial.read();

         if (y==250){
           throttle_gain=x[0];
           yaw_gain=x[1];
           pitch_gain=x[2];
           roll_gain=x[3];

           throttle_gain_last=throttle_gain;
           yaw_gain_last=yaw_gain;
           pitch_gain_last=pitch_gain;
           roll_gain_last=roll_gain;

           analogWrite(throttle,throttle_gain);
           analogWrite(yaw,yaw_gain);
           analogWrite(pitch,pitch_gain);
           analogWrite(roll,roll_gain);
           }
        }
```

```
    }
  else{    analogWrite(throttle,throttle_gain_last);
          analogWrite(yaw,yaw_gain_last);
          analogWrite(pitch,pitch_gain_last);
          analogWrite(roll,roll_gain_last);
           }
}
```